# MOBILE-BASED INDOOR NAVIGATION APPLICATION WITH OBJECT DETECTION FOR VISUALLY IMPAIRED

By

KUEK LEVIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

February 2026

# COPYRIGHT STATEMENT

© 2026 Kuek Levin. All rights reserved.

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Prof. Ts. Dr. Liew Soung Yue, for his invaluable guidance, support, and encouragement throughout the course of this project. His expert advice and insightful feedback greatly contributed to the progress of my work and helped me overcome the challenges I encountered along the way. I am truly grateful for his mentorship and dedication.

I would also like to extend my heartfelt thanks to my parents and family for their love, support, and continuous encouragement throughout this journey. Their unwavering belief in me has been a source of strength and motivation.

Finally, I wish to thank my friends for their support, understanding, and companionship. Their encouragement and presence have made this experience more meaningful and enjoyable.

# ABSTRACT

Currently, visually impaired individuals face significant challenges in achieving safe indoor navigation. Existing solutions are often impractical and not cost-effective. Therefore, this project proposes an indoor navigation mobile application to enhance the mobility and independence of visually impaired users. The system uses pedestrian dead reckoning (PDR) to estimate user movement and position. The YOLO model further enhances the system's capability for obstacle detection and object recognition. The integration of distance estimation into the YOLO model enables users to identify and avoid hazards more effectively, thereby enhancing navigation. By using the A* pathfinding algorithm, the system computes optimal routes and provides real-time guidance through auditory feedback. This solution aims to offer reliable, accessible, and user- friendly navigation assistance, improving safety and autonomy for visually impaired users in indoor environments.

Area of Study: Navigation, Object Recognition
Keywords: Visually Impaired, Indoor Navigation, YOLO, Object Detection, Distance Estimation

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| *$* | Dollar Sign |
| ± | Positive and Negative |
| *A** | A Star Algorithm |
| *cm* | Centimetre |
| *D** | D Star Algorithm |
| ° | Angle Degree |
| *ms* | Millisecond |
| *Wh* | Watt-hour (battery) |

# LIST OF ABBREVIATIONS

| | |
|---|---|
| *A** | A* algorithm |
| *AHRS* | Attitude and Heading Reference System |
| *AI* | Artificial Intelligence |
| *ALVU* | Array of Lidars and Vibrotactile Units |
| *API* | Application Programming Interface |
| *AR* | Augmented Reality |
| *D** | D* algorithm |
| *PDR* | Pedestrian Dead Reckoning |
| *FICT* | Faculty of Information and Communication Technology |
| *GPS* | Global Positioning System |
| *gTTS* | Google Text to Speech |
| *HMI* | Human-Machine Interaction |
| *IMU* | Inertial Measurement Unit |
| *IR* | Infrared Radiation |
| *JKM* | Department of Social Welfare |
| *LiDAR* | Light Detection and Ranging |
| *LRA* | Linear Resonant Actuator |
| *mAP* | Mean Average Precision |
| *P* | Precision |
| *QR* | Quick Response |
| *R* | Recall |
| *RFID* | Radio Frequency Identification |
| *RPI* | Raspberry Pi |
| *SLAM* | Simultaneous Localization and Mapping |
| *ToF* | Time-of-Flight |
| *TPU* | Tensor Processing Unit |
| *TTS* | Text to Speech |
| *UTAR* | Universiti Tunku Abdul Rahman |
| *UWB* | Ultra-Wideband |
| *VI* | Visually Impaired |

| | |
|---|---|
| *VIO* | Visual Inertial Odometry |
| *WHO* | World Health Organization |
| *Wi-Fi* | Wireless Fidelity |
| *YOLO* | You Only Look Once |

# CHAPTER 1
# Introduction

In this chapter, the background of indoor navigation for visually impaired users will be presented, together with the motivation behind this project. In addition, the specific problems it aims to address, and the main contributions of this work will also be discussed, including project objectives and scope.

## 1.1 Project Background

The field of indoor navigation has gained increasing attention in recent years due to its broad applications in surveillance, robotics, augmented reality (AR), and assistive navigation systems [1]. This surge in interest is driven by the increasing demand for precise and reliable positioning solutions in complex indoor environments. Among these applications, one of the most socially impactful lies in assisting visually impaired (VI) individuals.

Vision plays a crucial role in human navigation, which allows people to understand their surroundings and move safely through different environments. [2] For people with visual impairments, this fundamental ability is significantly reduced, which creates serious challenges in their daily mobility. According to the World Health Organization (WHO), at least 2.2 billion people are living with vision impairment in the world [3]. Narrowing it down to Malaysia, the Department of Social Welfare (JKM) reported that more than 65,000 individuals have been officially registered as visually impaired (VI) [3]. This significant number reflected the importance of recognizing the challenges they face in daily life. Since 80% to 90% VI individuals spend most of their time in indoor environments, the need for effective navigation systems designed specifically for spaces such as shopping malls, universities, and private residences is further highlighted [2].

Currently, most VI individuals still rely on the traditional assistive tools such as white canes and guide dogs [2]. Although these tools provide the essential basic support for mobility, they are limited in functionality. For instance, a white cane can detect obstacles within a certain range, but it is unable to deliver accurate spatial awareness.[2] Similarly, guide dogs are not easily accessible because they require extensive training

and involve high maintenance costs [2]. At the same time, they are prohibited from entering some indoor environments [2]. Therefore, it is still challenging for visually impaired people to navigate safely and independently.

To tackle these challenges, researchers have analysed and provided various technological solutions for supporting the navigation. The Global Positioning System (GPS) is commonly adopted in navigation field because it provides accurate and reliable location tracking for users [5], [6]. However, in indoor settings, weak satellite coverage and signal interference greatly diminish its effectiveness [5], [6]. Consequently, alternative methods such as Wi-Fi, Bluetooth Low Energy (BLE) beacons, and Radio Frequency Identification (RFID) have been proposed to overcome these limitations [5], [6]. While these technologies can operate where satellite signals are unavailable, they often demand extensive infrastructure, which makes them expensive and inappropriate for many public areas. Therefore, this project aims to design a low-cost indoor navigation mobile application for visually impaired individuals that delivers accurate guidance with minimal infrastructure to improve their accessibility and independence.

## 1.2 Problem Statement

Currently, VI individuals often face significant challenges when navigating indoor environments. While outdoor navigation technologies such as GPS are widely adopted, they become ineffective in indoor settings due to signal degradation and interference from the building structures [5], [6]. This limitation reduces the independence and safe navigation for VI people, which affect their autonomy and overall quality of life.

Existing indoor navigation systems are often expensive and fail to provide effective navigation or intuitive guidance tailored to the needs of VI individuals. Moreover, common assistive tools such as white canes and guide dogs provide only basic support to VI users [2]. However, these tools are unable to detect dynamic obstacles, deliver spatial awareness and perform route planning. This gap highlights the need for reliable and accessible indoor navigation technologies that can enhance mobility and independence for VI users. At the same time, the given solution must be affordable by the VI users and easy to deploy as most of them facing financial constraints.

## 1.3 Motivation

This project is driven by the need to enhance the mobility and independence of visually impaired (VI) individuals. While VI users often rely on traditional assistive tools such as white canes and guide dogs, these tools offer only limited functionality including basic obstacle detection and basic orientation [2]. As a result, many VI users experience become not confidence and not safety when navigating complex indoor environments.

To address these limitations, this project aims to develop an effective and cost-efficient indoor navigation mobile application tailored to the needs of VI users. By combining intuitive guidance and effective navigation, this project seeks to provide for VI individuals with greater autonomy in indoor environment to improve the quality of life.

## 1.4 Project Objectives

The primary objective of this project is to develop an indoor navigation mobile application that assists VI individuals in improving their mobility and independence. Instead of replacing traditional tools such as the white cane, the proposed system is designed to complement them by offering enhanced support in a cost-effective and efficient solution. The sub-objectives are:

- Provide better indoor navigation experience for visually impaired individuals.
  Similar to the existing work that have done, this system aims to provide a safe and effective navigation for VI users to move in an indoor environment. At the same time, it offers enhanced support alongside conventional tools.
- Design a lightweight system that is cost-effective and easy to use
  The project is developed to address the high cost and complexity of most existing products, making it more accessible and user-friendly for visually impaired (VI) users. By using built-in smartphone sensors for step detection and heading tracking, the system avoids the need for expensive external hardware.
- Integrate an object detection model for recognizing obstacles and objects.
  A fine-tuned object detection model is used to detect hazards and objects in the environment.
- Provide real-time feedback and instruction to users

The system can deliver guidance through audio feedback to guide users and alert them about the obstacles in real time.

## 1.5 Project Scope

The scope of this project is to design and develop an indoor navigation mobile application for visually impaired (VI) individuals. The applications will be implemented and tested in the meeting room (NF-023), Faculty of Information and Communication Technology (FICT), Universiti Tunku Abdul Rahman (UTAR). The prototype will be developed as a working mobile application that consists of indoor positioning, navigation, and HMI modules. The system will include the following functions:

- A navigation module that determines the optimal path to the destination.
- Implementation of a lightweight model for object detection and recognition of items.
- Object detection to identify the obstacles that are present in the environment.
- Receive instructions and provide guidance via audio feedback.

This project only focuses on developing a mobile application for indoor navigation within a controlled environment. It will not cover large-scale deployment across multiple buildings. Future enhancements may include improving the overall system in term of accuracy and reliable, refining object detection for higher accuracy.

## 1.6 Contributions

This project aims to develop an effective and affordable indoor navigation system to address the challenges faced by the VI users. The proposed solution will combine a range of technologies including pathfinding algorithms, object detection methods, sensory feedback mechanisms, and a robust system architecture to provide smooth and reliable navigation experience. The integration of the object detection model as a supporting feature enables the system to identify surrounding objects. This feature enhances environmental awareness and provides visually impaired (VI) users with more reliable and comprehensive navigation support.

In conclusion, this project demonstrates a strong potential for real-world application with the combination of technical feasibility and social impact. This contribution is

valuable to the field of assistance technology while improving accessibility and quality of life for the VI users. Over time, this initiative fosters increased social equality and supports the smoother inclusion of vulnerable groups within the broader community.

## 1.7 Report Organization

The report is divided into seven chapters, each presenting the details of the project and research. Chapter 1 provides an introduction to the project, which cover the background, problem statement, motivation, scope, objectives, contributions, and report organization.

Chapter 2 focuses on the literature review, including indoor localization and navigation techniques, distance estimation as well as object detection models. In addition, all techniques are reviewed to identify their limitations.

In chapter 3, this section discusses the method and approach that was used in the project. The system development methodology and system architecture are shown and explained in here. In addition, system architecture diagrams, use case diagram and activity diagrams are present in here too. Furthermore, the timeline of the project is also presented in this section.

For chapter 5, the implementation of the system is described with the related development information. This section explains the tools used, the hardware and software setup, and the development of the modules that form the core of the application. A short remark is concluded in the end of this section.

Chapter 6 talks about the evaluation and discussion of this project. It covers system testing and performance metrics, and the testing setup with corresponding results. The chapter also highlights project challenges, evaluates the objectives and conclude with a concluding remark.

For the last chapter, chapter 7 outlines the key achievements, project limitations and recommendations for future enhancement.

# CHAPTER 2

# Literature Reviews

This section reviews existing studies, products, and technical approaches related to indoor navigation systems for visually impaired (VI) individuals. The purpose is to identify the current state of research, evaluate the advantage as well as limitation of existing solutions.

## 2.1 Core Modules of Indoor Navigation System

In general, indoor navigation system is made up by three modules that are indoor positioning system module, navigation module and human-machine interaction (HMI) module, as illustrated in Figure 2.1.1.1 [5]. These modules work together to provide accurate positioning, route planning and effective communication for guidance. Therefore, a smooth and reliable navigation can be delivered to the VI users.

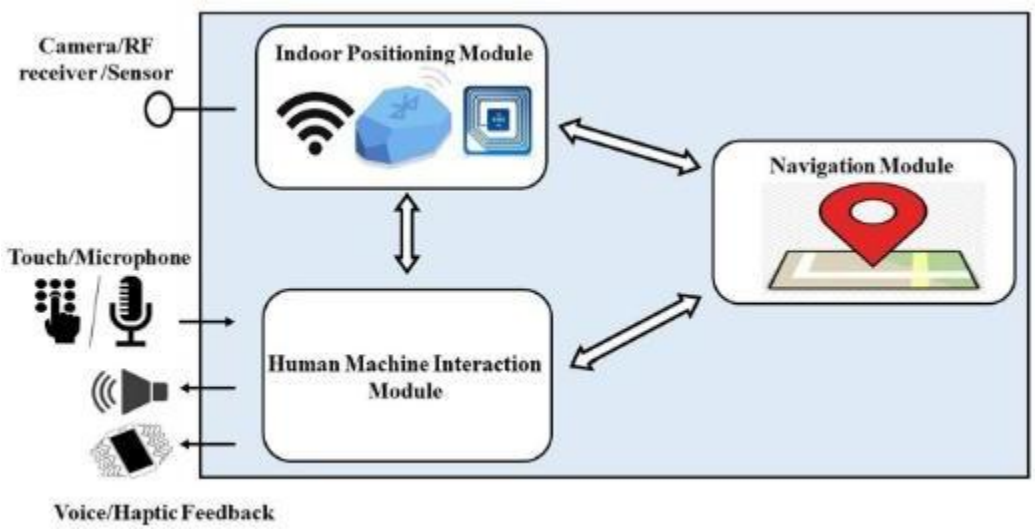



*Figure 2.1.1: Overall System Architecture of Indoor Navigation System in [5]*

Among the three core modules of indoor navigation system, indoor positioning system is the most critical, as it provides the foundation upon which navigation and HMI modules operate. It is responsible to identify the user's location in the indoor environment where the GPS signals are unavailable [5], [6]. The common assistance of technologies such as Wi-Fi, RFID and BLE beacons are used to address these issues. They aim to provide real-time localisation to support navigation, but each of them has presented trade-offs in term of accuracy, cost and deployment complexity.

Later, navigation module utilises user location data that provided by the previous module to determine an optimal path to direct VI users to their intended destination [5]. The pathfinding algorithms will evaluate the overall layout of the environment and consider factors such as potential obstacles and the shortest available routes. They are commonly graph search algorithms including A*, Dijkstra and D* algorithm [5].

After the routes have been computed, HMI module conveys navigational instructions through non-visual channels tailored to VI users [5]. By using audio and haptic feedback, it guides users by indicating direction changes, alerting them to obstacles, and confirming arrival at the destination [6]. In addition, the HMI module also receives user input when a new destination is requested.

This modular structure forms the backbone of most indoor navigation solutions. Building on this framework, previous studies have investigated a wide range of technologies to improve each module's performance, but many still fall short in term of affordability and accessibility for further innovation.

## 2.2 Previous Works on Indoor Navigation Technologies

### 2.2.1 The SUGAR System: UWB-Based Indoor Navigation Approach

In [7], the authors proposed SUGAR system to address the limitation of GPS which unable to operate in indoor environments as well as the previous existing solution's drawback. The system operates by continuously track the user's position through Ubisense Ultra-Wideband (UWB) sensors, which put on the wall and a UWB tag worn by the users. The user's heading is determined by the smartphone compass. After that, the positional and orientation data is mapped onto a spatial database derived from the building's floor plan, which is partitioned into navigable paths and obstructed zones. The navigation initiates when users select a desired destination to the smartphone from a predefined list, which is played audibly through user's headphone. By integrating with the A* pathfinding algorithm, the system computes an optimal path to user intended destination. At last, the navigation guides are delivered in audio form through headphone to guide the users. The interaction of these components is illustrated in Figure 2.2.1.1.

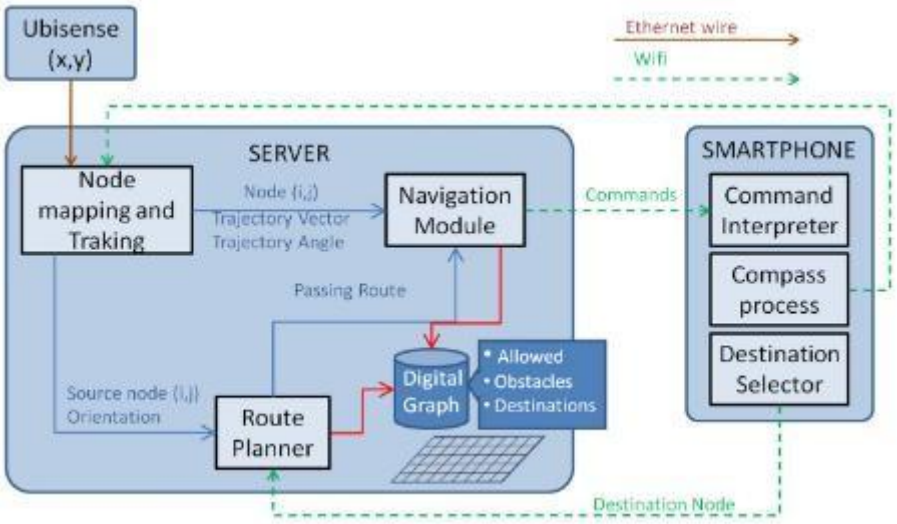



*Figure 2.2.1.1: System Architecture of SUGAR System in [7]*

UWB technology was chosen over RFID technology because it provides a larger operating range and higher positioning accuracy while consuming less power, which makes it requires fewer sensors to cover large area [5], [6], [7]. This reduction in hardware not only simplifies installation but also contributes to lower overall system cost. In addition, UWB can operate even there are obstacles blocking the signal path [7]. These advantages are demonstrated by the SUGAR system, which achieves positioning accuracy within 20cm across most locations [7]. According to [7], the authors used the A* algorithm as it offers the shortest path. For user guidance, the system integrates continuous tick-tock acoustic signals and voice prompts. This feedback mechanism is designed to minimise cognitive load and reduce uncertainty during navigation. Since VI users rely on auditory input, the system preserves one ear for environmental awareness by using a single earpiece. The authors also suggest the use of bone conducting headphones to prevent cover ear and potential discomfort during prolonged use.

### 2.2.2 Indoor Navigation System with QR Code and Text-to-Speech

Based on [8] study, they addressed the indoor navigation challenges faced by the VI individuals by providing an Android-based indoor navigation system. The system uses horizontal strips of QR codes placed on the floor for location detection, paired with a smartphone application that delivers audio guidance. Each QR codes contain location information that helps to determine user current position. Location tracking is achieved

by using the ZXing library to perform continuously scanning QR codes that placed along predefined indoor pathways. After VI users select a destination via a Text-to-Speech interface, the system retrieves the relevant map data from the SQLite database and computes the optimal path using pathfinding A* algorithms. The navigation instructions including corrective instructions when deviations detected, are delivered in audio format. The overall workflow is illustrated in Figure 2.2.2.1.

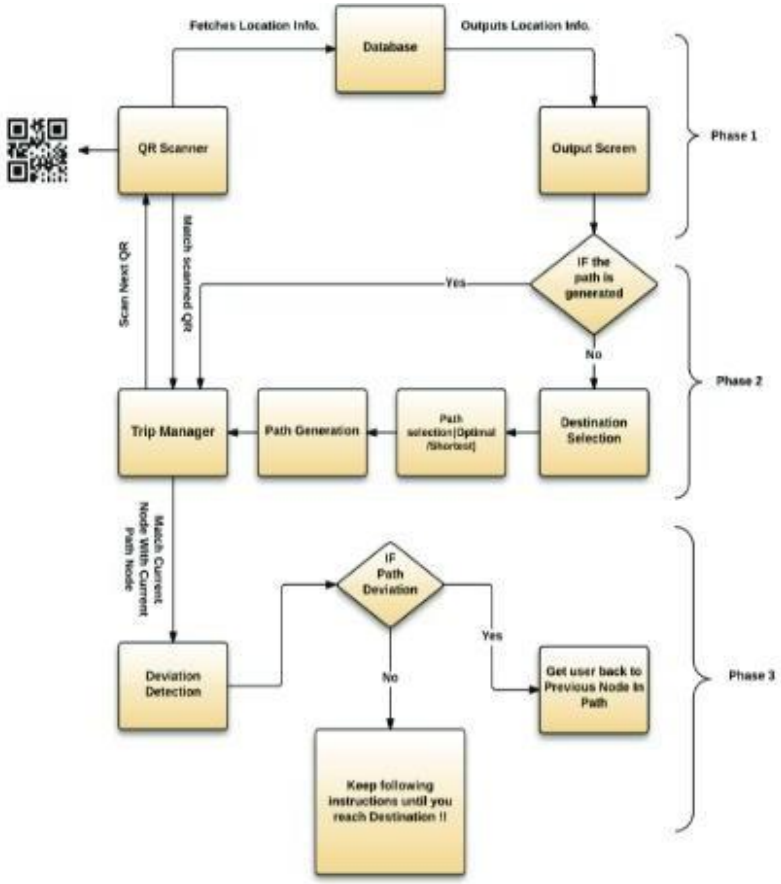



*Figure 2.2.2.1: Operational Diagram of Indoor Navigation System in [8]*

This approach offers a cost-effective and accessible solution that is able to work offline for VI users. The QR codes are utilised in this work because it offers benefits such as high data storage capacity, fast and omnidirectional readability, offering distinct benefits compared to other visual markers. They also eliminate the need of additional hardware as the system requires only a smartphone that is widely owned by most individuals. Moreover, the A* pathfinding is chosen due to its ability to balance

completeness and optimality. The integration of audio guidance and gesture-based interface further enhances accessibility and usability for VI users.

On the other hand, the system has several limitations. It depends on pre-installed QR codes, which restrict to predefined areas and requires regular maintenance to ensure the markers remain visible and undamaged. Since the codes are placed on floor, users must direct their smartphones downward to scan them, making the process inconvenient during movement. Although the system able to handle QR code with up to 60% blurriness, its scanning performance is influenced by factors such as lighting conditions, marker size, and marker clarity. The absence of obstacle detection and real time hazard awareness also limit the system's ability to ensure safe navigation especially in environments with unpredictable conditions.

### 2.2.3 Indoor Pathfinding Application with Obstacle Detection

In this previous work [9], the authors proposed a system that integrates deep learning with heuristic pathfinding to support indoor navigation for VI individuals. According to Figure 2.2.3.1, the system begins with a CCTV camera capturing video frames of the surrounding environment. The frame is segmented into multiple tiles, and each tile serves as a node within a grid-based representation of the navigation environment. The You Only Look Once (YOLO) v8 model is used to detect the obstacles within these frames. The blocked nodes will be marked as value 1, while the free nodes are marked as value 0. Based on the binary grid map, the system utilises A* pathfinding algorithm to compute the shortest and safest route from user's current location to the target destination by avoiding the nodes that marked as value 1. At last, the audio instructions are delivered to the user based on the computed path.

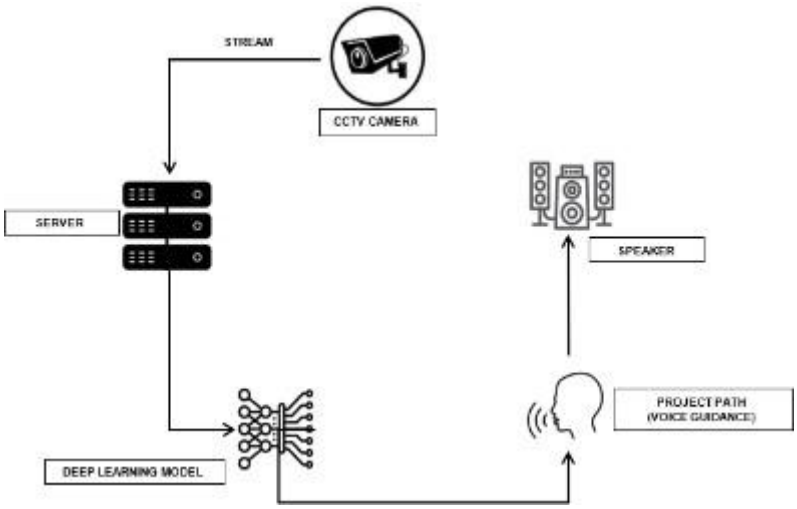



*Figure 2.2.3.1: System Architecture of Indoor Navigation System in [9]*

The system demonstrates several strengths. At first, the YOLOv8 model has achieved a high detection performance with near-perfect precision and 0.993 of mean Average Precision (mAP), which ensures high accurate obstacle identification. The system has real-time adaptability, allowing it to respond dynamically to environmental changes and maintain the most up-to-date path planning. The A* pathfinding algorithm is chosen for its efficiency and flexibility, as it can calculate the shortest path and adapt to various types of pathfinding problems through the use of a heuristic function. Moreover, this system supports diagonal pathfinding to ensure a more direct and efficient navigation path with fewer turns.

However, the system also faces some shortcomings. This system depends on the CCTV input, which limits environmental awareness to areas within the camera's field of view. The inaccurate mapping of obstacles can cause the A* algorithm to misclassify parts of objects as free space, leading to navigation paths that intersect obstacles and increase the risk of collision. When diagonal movement is disallowed, pathfinding results in longer, zigzag routes that can lead to unnecessary detours and increased travel time. Furthermore, current detection model is limited to identifying tables, people, and robots. Therefore, expanding the dataset is necessary to improve recognition of a broader range of object. The use of YOLOv8 model and real-time processing require high computational resources, which may be a challenge in term of power consumption and hardware scalability. The authors also emphasised the need for a better navigation

guide and voice-based destination input to enhance system functionality. In addition, speaker-based audio feedback may become inaudible in noisy environments.

### 2.2.4 The ALVU system: Time-of-Flight, IMU and Haptic Feedback

As discussed in [10], the authors proposed ALVU (Array of Lidars and Vibrotactile Units) system that ensures safe navigation for VI individuals by detecting obstacles at varying heights and recognising nearby physical boundaries. It is a wearable system that can contain two major components: a sensor belt and a haptic strap, as illustrated in Figure 2.2.4.1. The sensor belt is equipped with seven infrared time-of-flight (ToF) sensors to measure distances to the obstacles when user positioned around the waist. The belt also includes an Attitude and Heading Reference System (AHRS) to perform continuous estimation of absolute orientation by utilising data from inertial measurement units, including accelerometers, gyroscopes, and magnetometers. When obstacles are detected, the processed spatial information is delivered via Bluetooth to a haptic strap located on the user's upper abdomen. This strap contains five linear resonant actuator (LRA) vibration motors that deliver haptic feedback aligned with the sensor array's directional coverage. This enables VI users to perceive obstacle location and navigate safely.

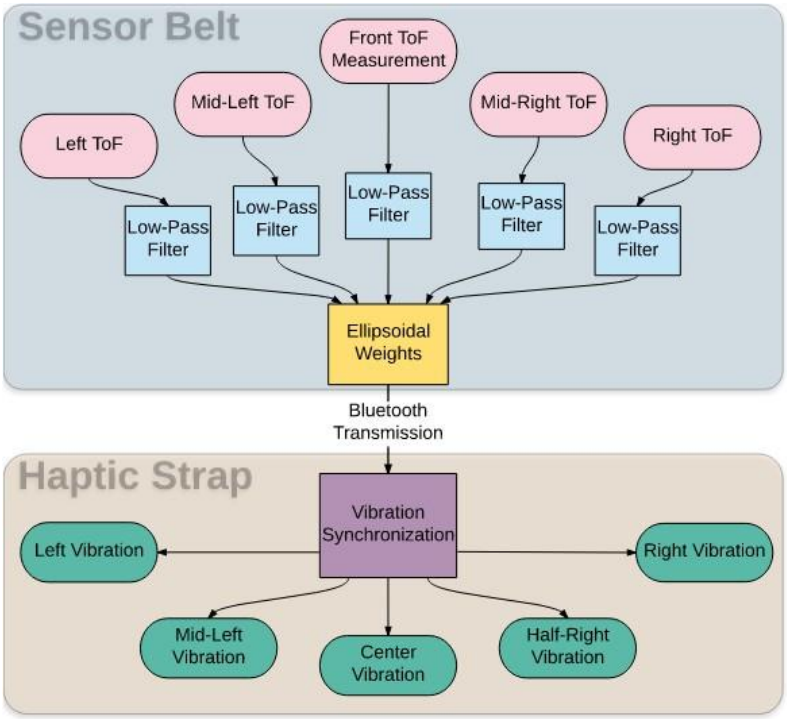



*Figure 2.2.4.1: System Architecture of ALVU System in [10]*

The ALVU system offers several strengths to assist VI users. Its hands-free and lightweight design enables users to focus on navigation without occupying their hands. The system is powered by 24.4 Wh lithium-ion battery, allowing the sensor belt to operate for up to 5.5 hours without recharging. The ToF sensor is selected to act as the sensor due to its ability to measure distances up to 14 meters, offering a resolution of 0.5 cm and an accuracy of 4 cm. The sensor belt can cover a ±70° horizontal field of view and ±45° vertical field of view of the user front view, which allows the system to detect obstacles at varying heights and provide more comprehensive environmental awareness than traditional canes. Moreover, the AHRS enables the system to distinguish between different terrain features such as obstacles, stairs, and elevation changes based on orientation and pitch analysis. The system provides the haptic feedback with approximately 120ms latency to support the obstacle alerts. For the haptic feedback, the system provides distinct vibration patterns that indicate different types of obstacles.

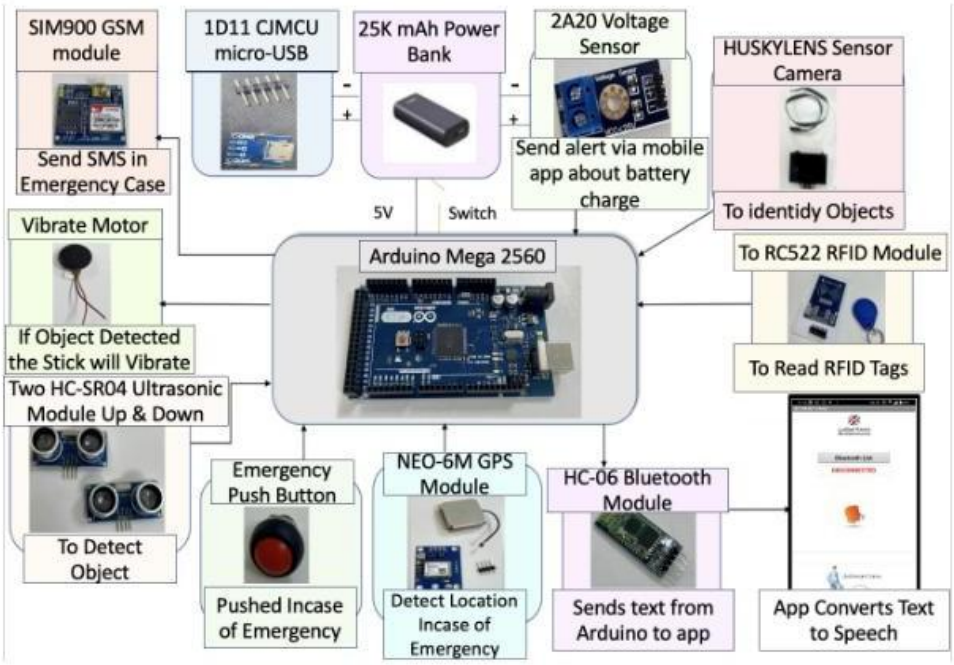
While ALVU system demonstrates its capability, but there are some drawbacks remain that may affect its performance. The sensor belt weighs 480 grams, which is about twice as heavy as a typical leather waist belt. When combining with the haptic strap, this may cause the VI users feel uncomfortable and bulky. Besides that, sensor belt's price is about $1300 due to the expensive cost of the ToF sensor. This creates another significant barrier as most of the VI users can't afford it. In addition, the system may struggle to detect obstacles approaching from extreme lateral angles. At the same time, the current system does not account for dynamically moving or irregularly shaped obstacles, especially in a complex environment. Therefore, the authors suggest that the system can be improved by integrating with adaptive viewing range adjustments based on the user's movement. Furthermore, [5] noted that the IMU is prone to cumulative errors and drift over time, resulting in degraded orientation and pitch estimation.

### 2.2.5 AI-Powered Smart Cane for Safer Navigation

[11] proposed an AI-powered smart cane that integrates multimodal sensing technologies to assist VI individuals in navigating an unfamiliar environment. As shown in Figure 2.2.5.1, this system uses an Arduino Mega 2560 as the core processing unit. It collects input from two HC-SR04 ultrasonic sensors and a HuskyLens sensor

camera for object detection. For localisation, the system utilises a NEO-6M GPS module for outdoor tracking and equips a RCS22 RFID reader to scan the tags that embedded in environmental objects for accurate indoor localization. By connecting the GPS module to a SIM900 GSM module, the system can send emergency messages containing location details to predefined caretakers when the SOS button is triggered by VI users. The processed guidance is transmitted from the Arduino Mega to the dedicated mobile application via a HC-06 Bluetooth module. This application converts text warnings into voice alerts and supplements them with tactile vibration cues triggered for nearby obstacles.

*Figure 2.2.5.1: System Architecture of Smart Cane in [11]*




The AI-powered smart cane demonstrates a few strengths that contribute to the safety navigation and user experience for VI users. Its foldable and lightweight design improves portability and user convenience. The large capacity of rechargeable lithium-ion battery increases the system's operational time. Furthermore, the authors chose ultrasonic sensor to be used in the system because it is cost-effective and energy-efficient compared to Light Detection and Ranging (LiDAR) and Infrared Radiation (IR) sensor. The HuskyLens sensor camera also was chosen due to its ability to recognise objects and process visual data. It also overcomes the ultrasonic sensor small detection range and extends the detection range up to 9 meters, which further enhances the object detection functionality. Moreover, the inclusion of RFID technology enables

reliable indoor localisation. By combining multiple sensors in the system, this can prevent single sensor failure risks. Additionally, SOS emergency messaging features adds an additional safety layer for VI users, while the multimodal feedback mechanism enhances situational awareness

On the other hand, this system also comes with some drawbacks. The system indoor positioning capability relies on pre-installed RFID tags, limiting the flexibility in deployment. In this study [11] does not explicitly state the type of RFID tags used. According to [5], [6], passive RFID tags can operate in 0.5 to 10 meters without any power source, which makes them cost-effective. The active RFID tags can achieve an operating range of up to 1 km, but they require internal battery and additional maintenance. Besides that, the effectiveness of ultrasonic sensor can be affected by the soft obstacles. Due to the short detection ranges of ultrasonic sensors, the system may depend on the HuskyLens sensor camera. As a result, the system's power consumption is increased. In addition, Figure 2.2.5.2 shows that the accumulation of hardware at the upper part of the cane may cause discomfort during use due to imbalanced weight distribution.


*Figure 2.2.5.2: Smart Cane Prototype in [11]*

### 2.2.6 IoT-BLE Based Indoor Navigation System

An IoT-BLE based indoor navigation system aimed at delivering real-time, accurate navigation for VI individuals was proposed in [12]. Based on Figure 2.2.6.1, the system starts by determining user's location once they enter the building. The BLE beacons installed throughout the building broadcast signals that can be detected by the user's device for localisation. The HC-SR04 ultrasonic sensors are used to capture information about the surroundings and detect obstacles. By combining the data from the BLE beacons and the ultrasonic sensors, the system creates a dynamic map according to the user's environment. User can switch the system to navigation mode via a dedicated mobile application. After inputting the desired destination via voice commands or accessible navigation interface, the system provides the real-time navigation guidance using multimodal feedback such as auditory, haptic, and visual cues. The system uses the Google Cloud Speech-to-Text API for voice recognition.

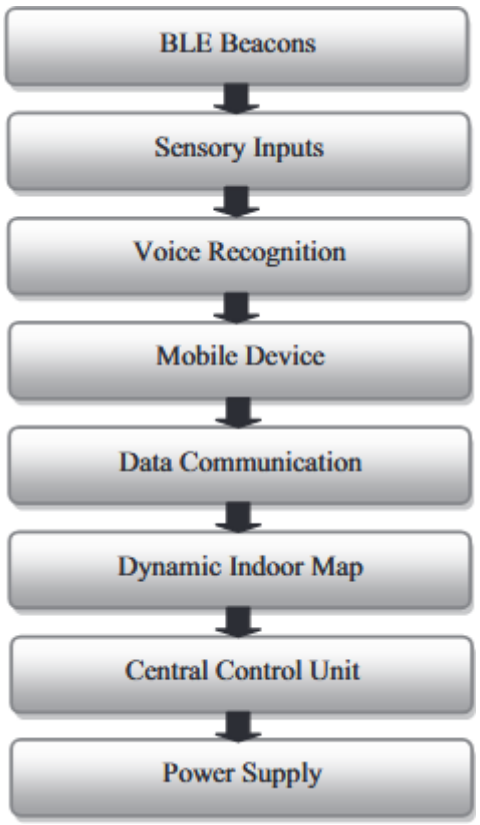



*Figure 2.2.6.1: System Workflow of Indoor Navigation System in [12]*

This IoT-BLE based indoor navigation system offers a few strengths. The system can provide real-time localisation by combining the BLE beacon technology with ultrasonic sensors. Furthermore, BLE technology is energy-efficient consumption and can operate over distances of up to 75 meters [5], [6], while ultrasonic sensors can detect obstacles within a range of 4 meters. The ability to continuously update the dynamic map ensures accurate and safe navigation. Besides that, the application's user interface features high-contrast text, voice command and gesture-based controls, which cater to the needs of VI users.

Despite these strengths, the system also presents some downsides. The accuracy and reliability of this system depend on the placement of the BLE beacons. Simultaneously, environmental factors like signal interference and the presence of certain building materials can weaken BLE signal strength and reduce positioning accuracy. Moreover, the ultrasonic sensors may have limitations in detecting soft obstacles as such obstacles can absorb high-frequency sound waves emitted by the sensors. Therefore, this reduces the effectiveness of the object detection. In addition, continuous data collection from users can lead to privacy and security concerns, as well as increased power consumption of user devices.

### 2.2.7 Camera Based Indoor Object Detection and Distance Estimation Framework

The authors proposed a framework to address indoor navigation challenges encountered by VI individuals by providing them with surrounding information and the relative distance to the objects [13]. According to Figure 2.2.7.1, this framework performs object detection by utilising a YOLOv7 deep learning model trained on an indoor object detection dataset about common household items. This model takes images or video frames as input and outputs a class label and bounding box for each detected object. The distance estimation is achieved by applying a bounding box-based formula, as illustrated in Equation 2.2.7.1. After that, the detection and distance data are converted into audio feedback through the Google Text to Speech Python library (gTTS) to provide informative navigation assistance to VI users.

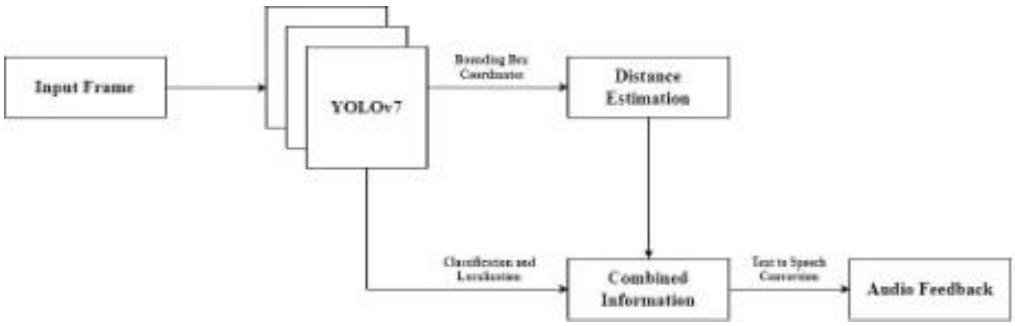



*Figure 2.2.7.1: System Architecture of Indoor Object Detection Framework in [13]*

$$D = \frac{2 \times 3.14 \times 180}{(w + h) \times 360 \times 1000 + 3} \quad (2.2.7.1)$$

where,D: Distance in inches

w: width of rectangle

h: height of rectangle

A key strength of this framework is its ability to deliver accurate and real-time object detection using the YOLOv7 model. The authors chose YOLOv7 over the YOLOv7-tiny model because the YOLOv7 model have better performance as indicated in the Table 2.2.7.1. Furthermore, the integrated distance estimation feature enhances spatial awareness. By providing continuous environmental information via audio feedback, the system remains accessible and practical for VI users.

| Model | Precision | Recall | mAP @.5 | mAP @.5:0.95 |
|---|---|---|---|---|
| YOLOv7 | 0.648 | 0.464 | 0.486 | 0.321 |
| YOLOv7-Tiny | 0.656 | 0.357 | 0.405 | 0.223 |

*Table 2.2.7.1: Model Performance Comparison Table [13]*

However, there are some limitations present in this framework. The dataset that was used to train the YOLOv7 model may lack sufficient coverage of the diverse objects present in indoor environments, potentially limiting recognition accuracy. Furthermore, the use of a deep learning model such as YOLOv7 may require high power consumption, which can limit portability and long-term usability. Moreover, the distance estimation feature provides only an approximate measure based on a single

camera. The dependence on monocular vision may be unreliable compared to multi-sensor systems. Besides that, single sensory cue for feedback may be inadequate to support the VI users.

### 2.2.8 Indoor Navigation Glasses with Deep Learning and Audio Guidance Using Google Coral Edge TPU

In [14], the authors developed an indoor navigation system for VI individuals through eyes glasses and a housing unit that focused on real-time vision processing and audio guidance. Initially, the system is activated via voice commands through an audio user interface that supports keyword recognition for functions such as localisation and navigation. When receiving location command, the MobileNetV2 deep learning model processes the live video feed from the Raspberry Pi (RPI) v2 camera to classify the user's current indoor environment. Once the navigation begins, the system provides sequential audio instructions, while the integrated MobileNet-SSD model performs obstacle detection within the camera's field of view. All processing is carried out on a Raspberry Pi 4 model B paired with the Google Coral Edge Tensor Processing Unit (TPU). enabling low-latency processing of visual and audio data to deliver seamless navigation assistance.

This system offers several significant strengths that enhance navigation for VI users. It contains two components, which are eyeglass and processing housing unit. The eyeglass contains RPi Camera v2, which ensure the design is wearable and lightweight, while the housing unit consists of the remaining components. The air vents design at the housing unit prevents overheating of the Raspberry Pi 4 Model B and Coral Edge USB Accelerator. Moreover, the system utilises Vosk model with the Kaldi recognizer toolkit for speech recognition. This is because they offer lightweight speech recognition and support for more than 20 languages. According to Table 2.1.9.1, the integration of MobileNet-based deep learning models allows for accurate localization, navigation, and obstacle detection achieves an overall classification accuracy of 97.4479% and an average response time of 0.6504ms. To prevent any room classification, a threshold of 0.7 is applied.

| Performance Evaluation | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Localization | 0.9188 | 1.0000 | 0.9188 | 0.9577 |
| Navigation | 0.9896 | 1.0000 | 0.9896 | 0.9948 |
| Obstacle Detection | 0.9958 | 0.9524 | 1.0000 | 0.9756 |
| Obstacle Avoidance | 0.9938 | 0.9512 | 0.9750 | 0.9630 |

*Table 2.2.8.1: Model Performance Table [14]*

Although the system shows many advantages, it also has some drawbacks that may affect its broad applicability. The camera's fixed field of view on the eyeglass may result in missed detections for obstacles located outside the frame. At the same time, the system is unable to detect dynamic obstacles that appear suddenly at lower levels, such as those near the user's legs. The use of deep learning models and real-time processing may also consume significant resources. Besides that, the headset used to provide audio feedback covers both ears, which is not ideal for VI users who rely heavily on auditory input. Therefore, the haptic feedback can serve as a valuable supplementary cue.

## 2.3 Limitation of Previous Studies

Previous studies on indoor navigation systems for visually impaired contributed valuable insights but they also present several limitations. A common challenge relates to the sensing and detection capabilities of these systems. For instance, [9], [13] suffer from limited fields of view, which may fail to detect the obstacles located outside the camera frame. In [11], [12], ultrasonic sensor faces reduced detection accuracy due to their short detection range and the presence of soft obstacles that absorb emitted sound waves. Furthermore, the IMU in [9] is prone to cumulative drift over time, causing errors for orientation and pitch measurements.

Moreover, several systems demonstrate a strong dependence on pre-installed infrastructure, such as BLE beacons, RFID tags and QR codes [7], [8], [11], [12]. This restricts the deployment to predefined areas and require regular maintenance to ensure consistent functionality. In addition, their performance can be affected by the environmental factors, including signal interference and building materials that attenuate wireless signals. As a result, this can reduce the positioning accuracy and

overall system reliability. System that depends on network connectivity may also suffer reduced reliability in areas with weak coverage [7].

The dataset used to train the deep learning model are insufficient to represent the diversity of objects that can found in indoor environments, which limits systems' object recognition capabilities [9], [13]. The absence of obstacle detection in [8] significantly increases the risk of collisions and potential harm to VI users. Besides that, the inability to detect dynamic obstacles especially those that appear suddenly, reducing the overall navigation safety for VI individuals.

For feedback mechanisms, [7], [8], [9], [13], [14] only support audio feedback can be problematic in noisy environments as the VI users may be unable to hear the cues clearly. In addition, the use of a headset that covers both ears may further reduce environmental awareness. This is because VI users rely heavily on surrounding sounds for orientation and navigation.

In terms of cost considerations, ToF sensors and UWB modules utilised in [7], [10] create affordability barriers for VI users. Additionally, the use of deep learning models and real-time processing introduces high reduce accuracy power consumption, making the systems unsuitable for long-term usability [9], [13], [14]. Furthermore, systems that involve continuous data collection can raise privacy and security concerns when the location and movement data is stored or transmitted.

## 2.4 Proposed Solutions

To overcome the limitations of existing indoor navigation systems, this project has proposed a mobile-based indoor navigation mobile application to provide safe indoor navigation for VI users. By utilising technology devices that most people already own such as smartphones, the system remains cost-effective and widely accessible. Therefore, VI users can move the mobile phone around to detect obstacles more effectively, thereby minimizing the risk of missed detections and enhancing overall navigation safety. In addition, this system can perform Pedestrian Dead Reckoning (PDR) with IMU data, enabling the system to estimate the user's position by incrementally updating location based on detected steps and orientation changes. To address the cumulative drift in IMU, the system performs cross-checking with other

sensors to improve the accuracy of orientation data. The system prioritises local processing to safeguard user privacy and security

In terms of object detection, a pretrained deep learning model such as YOLO is selected for object recognition and obstacle detection. The selected model must be energy-efficient and lightweight to minimise battery consumption on the phone and ensure long-term usability. Furthermore, the training dataset should be large enough to cover the range of objects commonly found in indoor environments, which supports reliable detection performance.

From the feedback perspective, the system provides multimodal feedback through audio cues. For audio feedback, the use of bone conducting or single-ear earphones are preferred as they allow VI users to maintain awareness of surrounding sounds that are critical for safe navigation.

# CHAPTER 3

# System Methodology/Approach

The processes of the project were categorized into different phases in the development, which were project pre-development, data pre-processing, model training architecture building and data training, and prediction on test dataset.

## 3.1 Development Methodology

This project will adopt an incremental and iterative Agile development methodology. The approach is chosen due to its flexibility and adaptability. It enables rapid adjustments based on user feedback, supporting continuous improvement and progressive refinement of the system. Moreover, this method allows the system to build and test each functional component, which helps to identify and address the issues in early stages. By breaking the project into chunks of manageable tasks, it helps maintain steady project progress and prevents task overload. Additionally, Agile emphasises prioritisation to ensure that the most critical tasks are addressed first, which makes it ideal for project with tight time constraints.

### 3.1.1 Methodologies and General Work Procedures

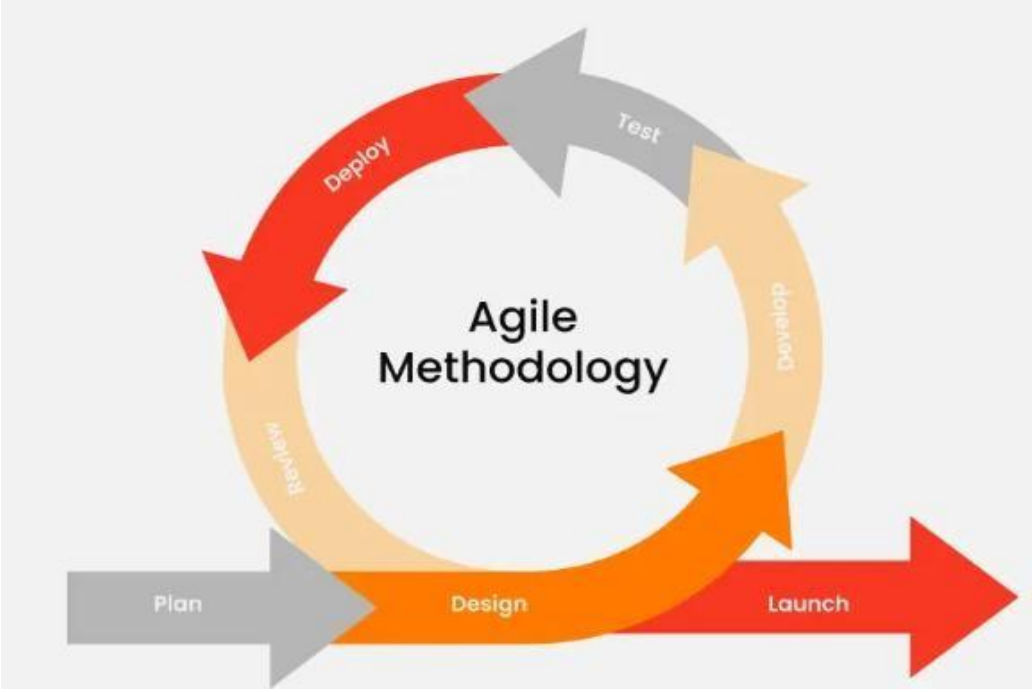



*Figure 3.1.1.1:* Agile Methodology

1. **Planning**

   The system requirements are defined by analysing the requirements and difficulties encountered by the VI users through this research. The necessary and appropriate hardware and software components are also considered to ensure feasibility and cost-effectiveness.

2. **Design**

   The system architecture and design are developed to serve as the backbone of the project. This includes data flow diagram, activity diagram and entity-relationship diagram to guide the following development process.

3. **Development**

   In this phase, the planned features are implemented incrementally. Each feature is developed as a small, functional unit that can be quickly tested and evaluated.

4. **Testing**

   Testing happens continuously alongside development. Each increment is validated through unit tests, integration tests, or user acceptance testing to ensure it meets the requirements and works without any bugs or errors.

5. **Deployment**

   Once features pass testing, the system is deployed in the intended environment for real-world operations. Deployment is done frequently to deliver value early and configure the system for the target setting.

6. **Review**

   During this phase, the feedback is collected from the users to identify areas for improvements and ensure the system aligns with the system requirements and user expectations.

7. **Launch**

   The final version of the system is released for end users to use. Documentation and support resources are provided to facilitate adoption and long-term operation.

There will be an iterative cycle from the design phase to the review phase enables the system to evolve and improve through incremental development and iterative feedback.

## 3.2 System Design Diagram/Equation

### 3.2.1 System Architecture Diagram

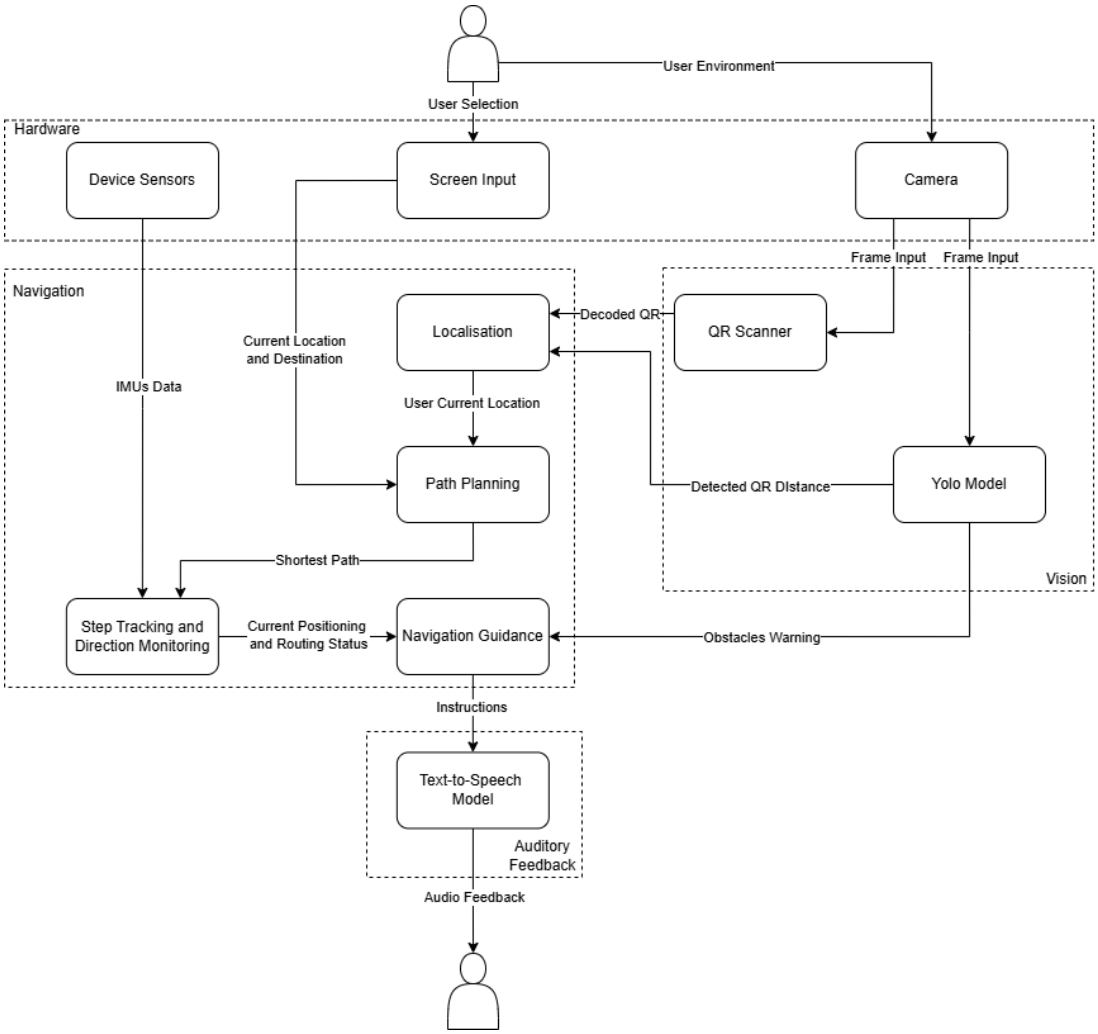



*Figure 3.2.1.1:* System Architecture Diagram

The Figure 3.2.1.1 shows the system architecture of the mobile-based indoor navigation application. The hardware components support both user interaction and environmental sensing. The screen functions as the primary interface that allow users to specify their current location and desired destination. At the same time, the mobile camera continuously captures frames from the surrounding environment. In addition, the device sensors including the accelerometer, gyroscope, and magnetometer, collect motion and orientation data, which are essential for determining the user's heading and step count.

Next, the fine-tuned YOLOv8n model processes the captured frames, identifying objects and estimating their distances. Meanwhile, the QR scanner decodes all QR

codes detected in the frames. Both the decoded QR code information and the corresponding distance estimates from the YOLO model are forwarded to the localisation. All non-QR detected objects are considered obstacles, and their distance estimates are transformed into obstacle warnings that are passed to the text-to-speech model for auditory feedback.

After that, the path planning module determines the shortest path by integrating the user's current location and desired destination, obtained from the localisation component and screen input. The step tracking and direction monitoring module then combines the planned route with IMU sensor data to monitor the user's movement, track steps, and determine heading direction. This information is passed to the navigation guidance module, which generates real-time routing instructions. Finally, these instructions are delivered to the text-to-speech model, which converts them into auditory feedback. This ensures that the user receives clear and continuous navigation guidance.

### 3.2.2 Use Case Diagram

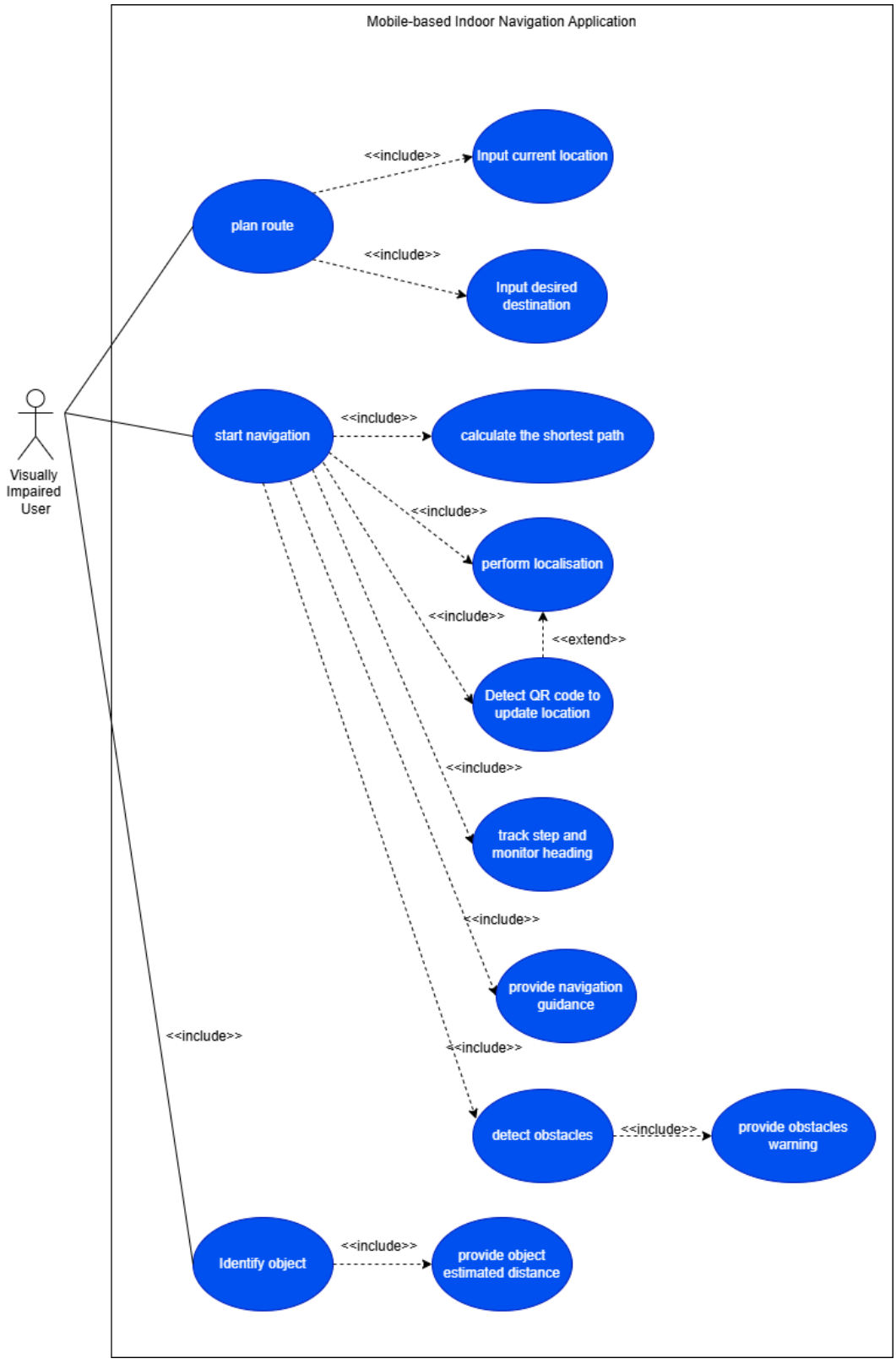



*Figure 3.2.2.1: Use Case diagram*

The use case diagram in Figure 3.2.2.1 illustrates the interaction between the VI user and the mobile-based indoor navigation application. In this diagram, the VI user acts as the main actor, directly engaging with the system to achieve navigation support and environmental awareness.

The system provides three main functionalities: route planning, navigation execution, and object identification. For route planning, the user initiates the Plan Route use case, which includes selecting the current location and the desired destination. Based on these inputs, the system prepares the navigation path.

During navigation, the Start Navigation use case is activated. This process includes calculating the shortest path, performing localisation, detecting QR code tracking user steps as well as heading, and providing navigation guidance. At the same time, the system enhances user safety by detecting obstacles and providing auditory warnings during navigation. Furthermore, the localisation process can be extended by detecting QR codes to dynamically update the user's current location. In addition, the system provides a function that enables the user to identify detected objects within the environment and displays the estimated distance to objects.

In conclusion, the use case diagram demonstrates how the actor interacts with the system through three core functionalities, which are route planning, navigation execution, and object identification. By integrating these use cases, the diagram conveys the system's comprehensive support for indoor navigation.

### 3.2.3 Use Case Description

This section provides detailed descriptions of the three use cases. Each use case descriptions defines the system functionality from the user's point of view by describing the sequence of interactions between the user and the system required to accomplish specific tasks.

**UC001 – Plan Route**

| **Use Case ID** | UC001 | **Version** | 1.0 |
|---|---|---|---|
| **Use Case** | Plan Route | | |
| **Purpose** | To allow the VI users to input their current location and their | | |

| | | |
|---|---|---|
| | desired destination. | |
| **Actor** | VI user | |
| **Trigger** | VI user double taps on the Home screen of Navigation page | |
| **Precondition** | 1. VI user is on the Home screen.<br>2. The map and POI files are successfully loaded into the system. | |
| **Scenario Name** | **Step** | **Action** |
| | 1 | System announces the instructions to use Navigation page. |
| | 2 | VI user double taps on the phone screen. |
| | 3 | System detects gesture motion that is inputted by VI user. |
| | 4 | System displays a list of locations. |
| | 5 | System announces "Please select your current location." |
| | 6 | VI user swipes left and right to browse the available locations. |
| | 7 | System detects gesture input and announces the name of the currently focused location. |
| | 8 | VI user double taps to confirm the selected start location. |
| | 9 | System detects gesture input and saves the selected start location. |
| | 10 | System displays a list of destination. |
| | 11 | System announces “[Location Name] is selected as start location. Please select your destination.” |
| | 12 | VI user swipes left and right to browse the available destination. |
| | 13 | System detects gesture input and announces the name of the currently focused destination. |
| | 14 | VI user double taps to confirm the selected destination. |
| | 15 | System detects gesture input and saves the selected destination. |
| | 16 | System announces “[Location Name] is selected as destination. Starting navigation.” |
| **Alternative Flow - Navigate to** | 2.1 | VI user swipes left or right on the phone screen. |

| | | |
|---|---|---|
| **Recognition page** | | |
| | 2.2 | System detects gesture motion that is inputted by VI user. |
| | 2.3 | System redirects VI user to the Recognition page. |

*Table 3.2.3.1 Use Case Description for Use Case 1 – Plan Route*

**UC002 – Start Navigation**

| **Use Case ID** | UC002 | **Version** | 1.0 |
|---|---|---|---|
| **Use Case** | Start Navigation | | |
| **Purpose** | To guide the VI user safely to their destination with auditory navigation instructions. | | |
| **Actor** | VI user | | |
| **Trigger** | The system completes the "Plan Route" use case. | | |
| **Precondition** | 1. The map and POI files are successfully loaded into the system.<br>2. The start location and desired destination are successfully set.<br>3. Device camera and IMU sensors permissions are granted. | | |
| **Scenario Name** | **Step** | **Action** | |
| | 1 | System activates the device camera and IMU sensors. | |
| | 2 | System redirects VI user to the navigation screen. | |
| | 3 | System prompts the VI user to scan two reference QR codes. | |
| | 4 | VI user scans two QR codes that are located nearby. | |
| | 5 | System decodes and validates the scanned QR code against the POI files. | |
| | 6 | System extracts the exact coordinates of QR code from the validated QR code. | |
| | 7 | System overrides the start location with the exact coordinates. | |
| | 8 | System calculates the shortest path to the destination based on the map files. | |
| | 9 | System announces the navigation instruction to VI user. | |
| | 10 | VI user starts to walk based on the instructed path and input camera frame. | |

| | | |
|---|---|---|
| | 11 | System continuously tracks the user's steps and heading during navigation. |
| | 12 | System continuously processes camera frames to scan for obstacles and QR code during navigation. |
| | 13 | System updates navigation nodes. |
| | 14 | System determines that the current navigation node is the last node. |
| | 15 | System announces "You have arrived at your destination." when VI user reaches the destination. |
| | 16 | System stops tracking and releases all internal sensors and device camera. |
| **Sub Flow – Detect Valid Step** | 11a.1 | System continuously tracks the user's steps. |
| | 11a.2 | System validates user's steps as valid step. |
| | 11a.3 | System updates this current user's step to the total number of user steps |
| | 11a.4 | Return to Main Flow 13. |
| **Sub Flow – Detect Correct Heading** | 11b.1 | System continuously tracks the user's heading. |
| | 11b.2 | System validates user's heading as correct heading. |
| | 11b.3 | Return to Main Flow 13. |
| **Sub Flow – Detect Obstacles** | 12a.1 | System continuously processes camera frames to scan for obstacles during navigation. |
| | 12a.2 | System detects obstacles in camera frames. |
| | 12a.3 | System calculates the estimated distance to the obstacles |
| | 12a.4 | System provides obstacles warning. |
| | 12a.5 | System updates the navigation instructions. |
| | 12a.6 | Return to Main Flow 9. |
| **Sub Flow – Detect QR Code** | 12b.1 | System processes camera frames to scan for QR code during the navigation. |
| | 12b.2 | System decodes and validates the scanned QR code against the loaded POI files. |
| | 12b.3 | System extracts the exact coordinates of QR code from the validated QR code. |

| | | |
|---|---|---|
| | 12b.4 | System overrides the current estimated location with the exact coordinates. |
| | 12b.5 | Return to Main Flow 13. |
| **Alternative Flow – Detect Unknown QR Code On Start** | 4.1 | VI user scans two unknown QR codes. |
| | 4.2 | System announces "Invalid QR code detected. Please rescan." |
| | 4.3 | System decodes and validates the scanned QR code against the POI files. |
| | 4.4 | Return to Main Flow 3. |
| **Alternative Flow – Detect Invalid Step** | 11a.2.1 | System validates user's steps as invalid step. |
| | 11a.2.2 | System ignores this current user's step. |
| | 11a.2.3 | Return to Main Flow 13. |
| **Alternative Flow – Detect Incorrect Heading** | 11b.2.1 | System validates user's heading as incorrect heading. |
| | 11b.2.2 | System provides direction warning. |
| | 11b.2.3 | System updates the navigation instructions. |
| | 11b.2.4 | Return to Main Flow 9. |
| **Alternative Flow – Detect Unknown QR Code During Navigation** | 12b.1.1 | System processes camera frames to scan for unknown QR code during the navigation. |
| | 12b.1.2 | System decodes and validates the scanned QR code against the loaded POI files. |
| | 12b.1.3 | System ignores the unknown QR code without changing the current estimated location. |
| | 12b.1.4 | Return to Main Flow 13. |
| **Alternative Flow – Detect Not Last Navigation Node** | 14.1 | System determines that the current navigation nodes is not the last node. |
| | 14.2 | System updates navigation screen details. |
| | 14.3 | System updates navigation instruction. |

| | | |
|---|---|---|
| | 14.4 | Return to Main Flow 9. |

*Table 3.2.3.2 Use Case Description for Use Case 2 – Start Navigation*

### UC003 – Identify Object

| **Use Case ID** | UC003 | **Version** | 1.0 |
|---|---|---|---|
| **Use Case** | Identify Object | | |
| **Purpose** | To identify the object in the VI user's environment. | | |
| **Actor** | VI user | | |
| **Trigger** | VI user double taps on the Home screen of Recognition page. | | |
| **Precondition** | 1. VI user is on the Home screen.<br>2. Device camera permissions are granted. | | |
| **Scenario Name** | **Step** | **Action** | |
| | 1 | System announces the instructions to use Recognition page. | |
| | 2 | VI user double taps on the phone screen. | |
| | 3 | System detects gesture motion that is inputted by VI user. | |
| | 4 | System activates the device camera. | |
| | 5 | System redirects VI user to the recognition screen. | |
| | 6 | VI user inputs camera frame. | |
| | 7 | System continuously processes camera frames to scan for objects in the VI user's environment. | |
| | 8 | System calculates the estimated distance to the object. | |
| | 9 | System announces the object's name along with its estimated distance. | |
| | 10 | System displays the object's name and estimated distance on the screen. | |
| | 11 | VI user long presses on the phone screen to stop object identification. | |
| | 12 | System stops and releases device camera. | |
| | 13 | System redirects VI user back to the Home screen of Recognition page. | |
| **Alternative Flow - Navigate to** | 2.1 | VI user swipes left or right on the phone screen. | |

| | | |
|---|---|---|
| **Navigation page** | | |
| | 2.2 | System detects gesture motion that is inputted by VI user. |
| | 2.3 | System redirects VI user to the Navigation page. |

*Table 3.2.3.3 Use Case Description for Use Case 3 – Identify Object*

### 3.2.4 Activity Diagram

This section presents three activity diagrams: Plan Route, Start Navigation, and Identify Object. These diagrams illustrate the key flows of the project's use cases.

## UC001 – Plan Route

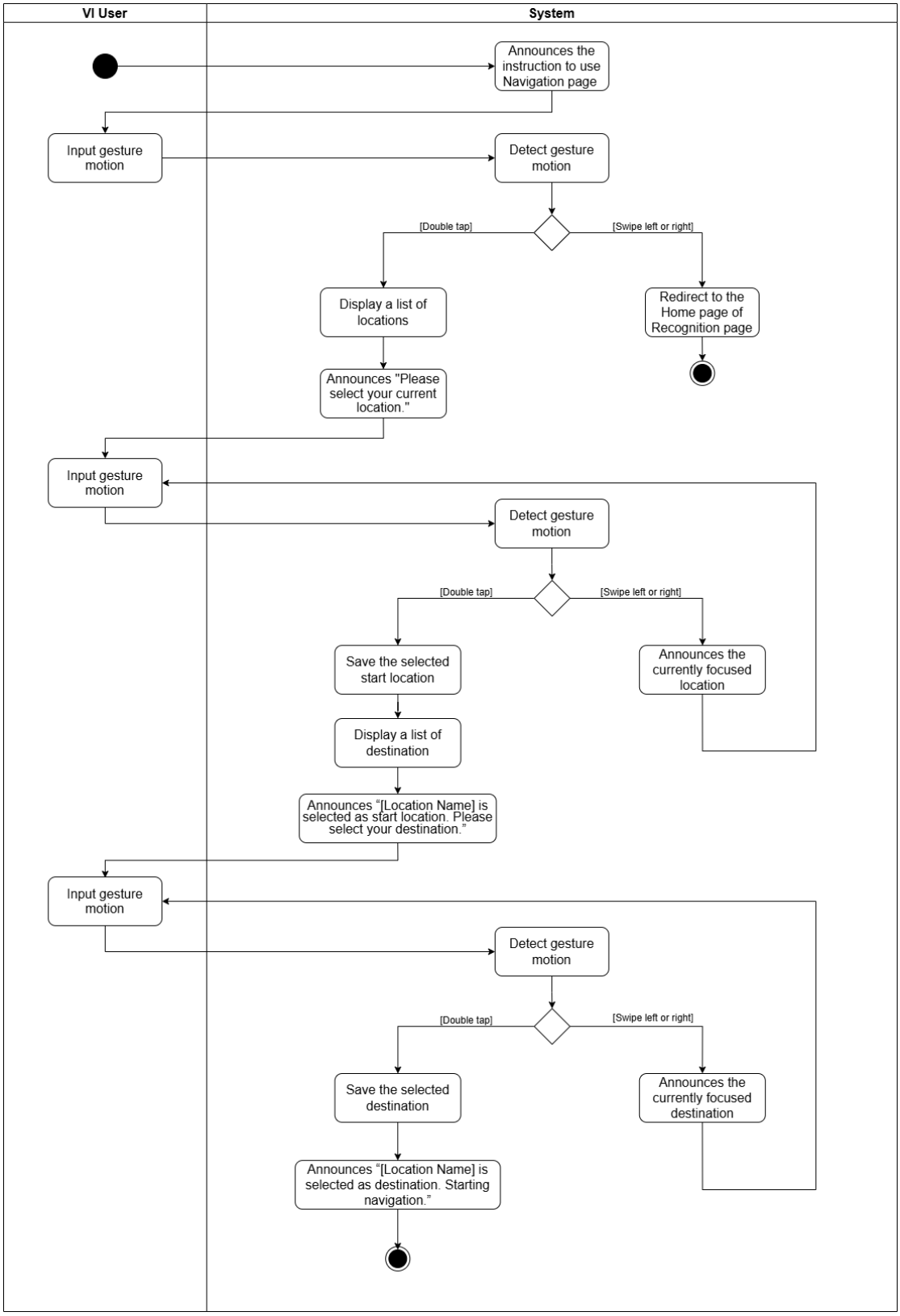



*Figure 3.2.4.1: Activity Diagram for Use Case 1 – Plan Route*

## UC002 – Start Navigation

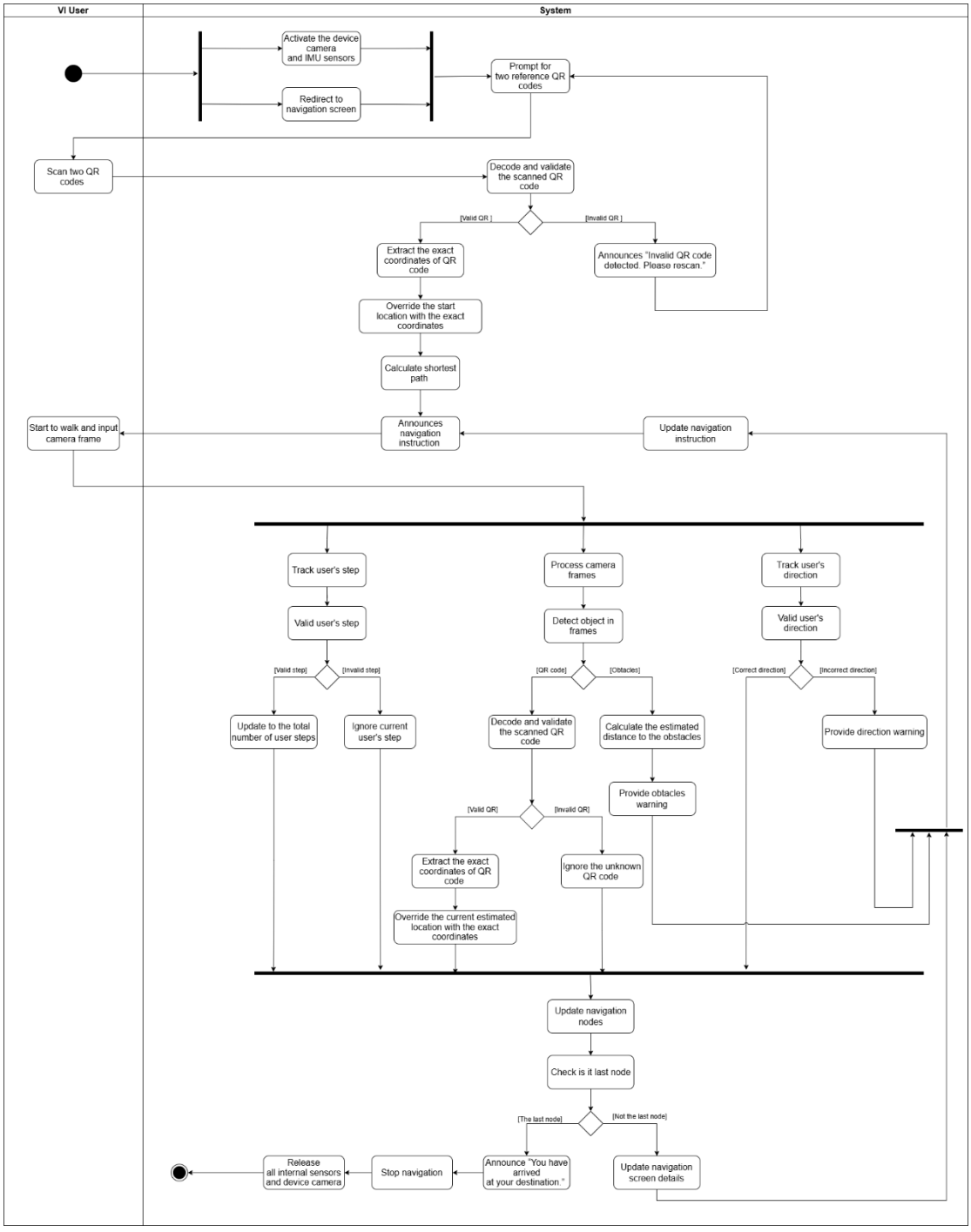



*Figure 3.2.4.2: Activity Diagram for Use Case 2 – Start Navigation*

## UC003 – Identify Object

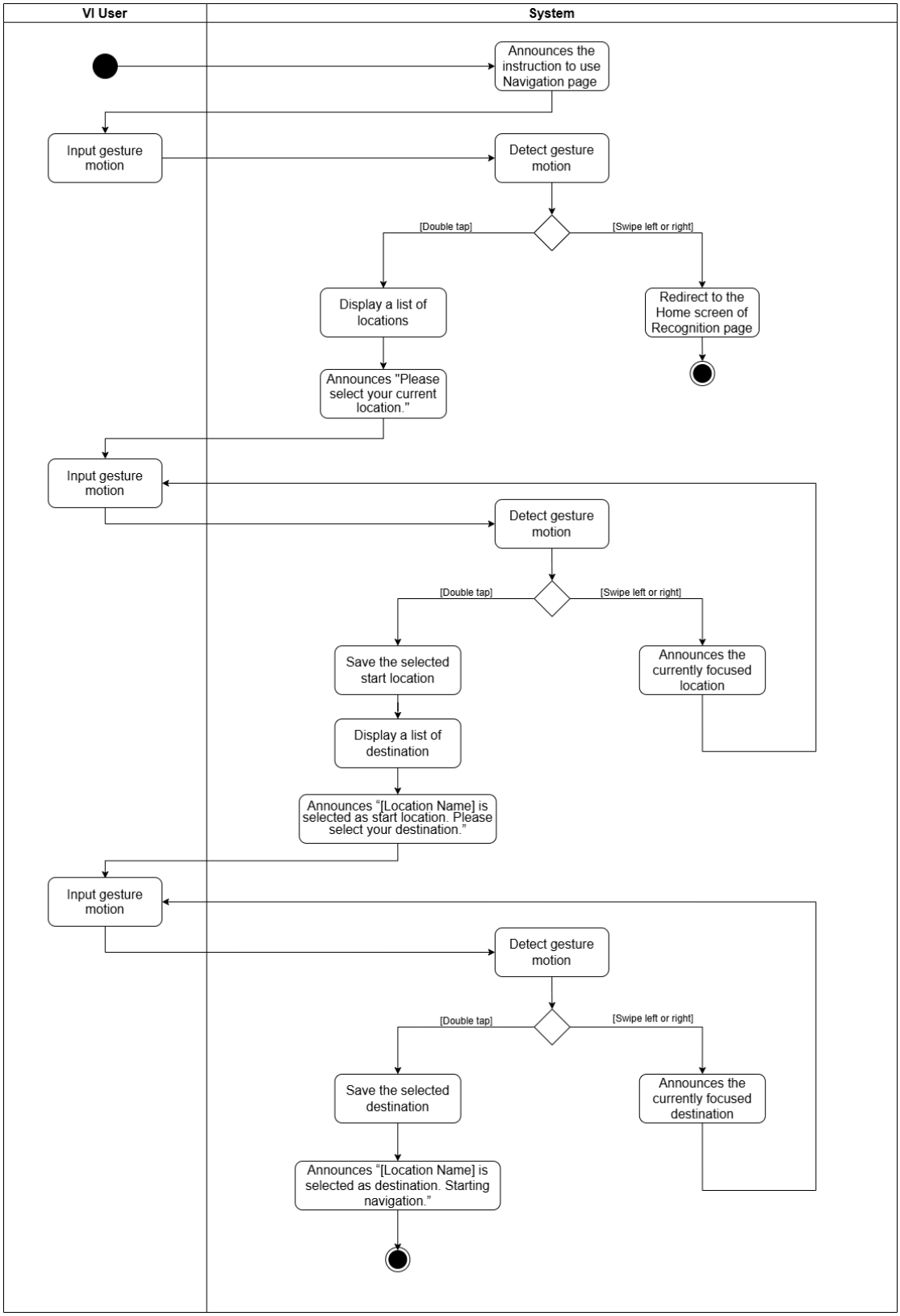



*Figure 3.2.4.3: Activity Diagram for Use Case 3 – Identify Object*

## 3.3 Project Timeline

This project is divided into two stages, which includes Final Year Project 1 and Final Year Project 2. Both stages are scheduled to be completed within two semesters. Both project timelines were presented in the Gantt chart. Figure 3.3.1 shows the project timeline for Final Year Project 1 over seven weeks, which emphasises prototype development. while Figure 3.3.2 presents the project timeline for Final Year Project 2 over 14 weeks, which focuses on refining prototype and completing the final deliverable. These charts facilitate progress tracking and support effective time management.

| Task Name | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 |
|---|---|---|---|---|---|---|---|
| Project Research and Analysis | | | | | | | |
| Project Planning and Design | | | | | | | |
| Prototype Implementation and Development | | | | | | | |
| Testing and Debug | | | | | | | |
| Report Writing | | | | | | | |
| Presentation | | | | | | | |

*Figure 3.3.1 Final Year Project 1's Timeline*

| Task Name | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | Week 11 | Week 12 | Week 13 | Week 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Work Planning | | | | | | | | | | | | | | |
| Dataset Selection | | | | | | | | | | | | | | |
| Model Training | | | | | | | | | | | | | | |
| Feature Enhancement | | | | | | | | | | | | | | |
| Feature Integration | | | | | | | | | | | | | | |
| Testing and Debug | | | | | | | | | | | | | | |
| Real Environment Testing and Optimisation | | | | | | | | | | | | | | |
| Report Writing | | | | | | | | | | | | | | |
| Presentation | | | | | | | | | | | | | | |

*Figure 3.3.2 Final Year Project 2's Timeline*

# CHAPTER 4

# System Design

## 4.1 System Block Diagram

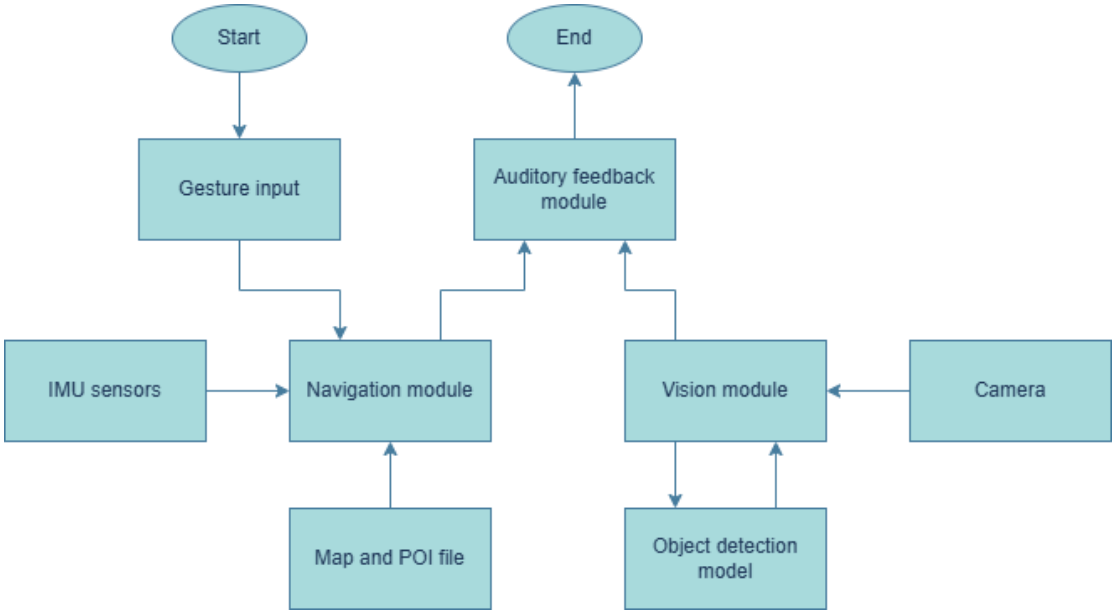



*Figure 4.1.1 System Block Diagram*

The system architecture of the indoor navigation application is designed as a modular to ensure high performance, maintainability and seamless data flow. As illustrated in Figure 4.1, this architecture categorises the system components into four distinct layers:

- Presentation layer: Consists of the gesture input and the auditory feedback Module, which handles direct interaction with the VI user.
- Core processing layer: Contains the navigation module and the vision module that act as the central model that processes logic, algorithms, and real-time path finding.
- Hardware and sensor layer: Comprises the IMU Sensors and the camera, responsible for capturing real-time physical and environmental data.
- Data and storage layer: Includes the map and POI file along with the object detection model to provide the necessary offline data resources to support the core processing layer.

The process begins when the user selects a current location and destination via gesture input. After selection, the system performs scanning of two QR codes in the

surrounding environment to support localisation. Once calibrated, the system utilizes the A* algorithm within the navigation module to compute the optimal shortest path to the destination.

As the user begins to walk, the navigation module and the vision module operate concurrently to ensure safe navigation while continuously detecting potential obstacles and QR codes. During navigation, navigation module continuously updates the user's real-time position and steps. Throughout this process, the system delivers step-by-step auditory navigation instructions via text-to-speech (TTS).

## 4.2 System Component Specification

In this indoor navigation application, three main screens are specifically implemented to support VI users through non-visual cues. These screens provide intuitive interaction methods that enhance accessibility and usability.

1. **Home Screen**

   This screen serves as the entry point of the application. It is designed with an intuitive and minimalist layout to prevent cognitive overload for VI users. Its primary purpose is to provide access to both the object recognition and navigation feature.

   **Accessibility Features:**

   - Gesture input interaction supports swiping actions to change modes and selecting the current location and destination.
   - Auditory feedback that delivers tutorial guidance, system status information and confirmation messages to ensure accessibility for visually impaired users.

2. **Navigation Screen**

   This is the core screen that designed for route guidance and movement tracking. It provides real-time navigation instructions and concurrently analyses camera frame input for obstacle avoidance to enhance user safety and support reliable indoor navigation.

   **Accessibility Features:**

   - Support step tracking and direction monitoring.

- Analyses continuous camera frame input to detect surrounding obstacles.
- Auditory feedback that delivers navigation guidance and obstacle warning.

3. **Recognition Screen**

   This screen allows VI users to scan their surroundings objects in order to better understand their environment.

   **Accessibility Features:**

   - Analyses continuous camera frame input to detect surrounding objects along with their estimated distance.
   - Auditory feedback that delivers navigation guidance and obstacle warning.
   - Gesture input interaction for terminating the process and returning to the previous screen.

## 4.3 Accessibility Design

Since the primary target actor is the VI user who lacks visual capability, the accessibility design of this indoor navigation application should not rely on visual cues. Therefore, all system functionalities are designed to emphasise on auditory feedback and gesture-based navigation.

### 1. Auditory Feedback

As VI user cannot view the digital map on screen, text-to-speech (TTS) serves as a crucial output channel of the system. The auditory feedback provides three types of announcements to ensure that the VI user remains aware of the surroundings and system status.

- **Navigational guidance:** Provides real-time and clear step by step navigation instructions that calculated by A* algorithms.
- **Obstacle warnings:** When obstacles are detected in dangerous proximity by the YOLOv8n model in the vision module, the system immediately interrupts the current audio broadcast to deliver obstacle warnings.
- **System status:** Informs the VI user about the current state of the application such as during the QR code scanning phase.

### 2. Gesture-based Navigation

On most of the traditional mobile application interfaces, they rely on small button and sight-based interaction which are inaccessible to VI user. In order to overcome this issue, the application introduces gesture-based input.

- **Non-visual interaction:** VI user can perform intuitive gestures input anywhere on the screen instead of locating the UI buttons. This approach reduces the learning curve and enhances usability. The gesture set includes tap, long press, vertical and horizontal swipes.

# CHAPTER 5
# System Implementation

## 5.1 Tools to Use

### 5.1.1 Hardware Specifications

The hardware for this project includes a laptop, an Android phone and a TurtleBot 3. The laptop is used for Flutter app development, dataset preparation, and object recognition model training, while the Android phone is used for testing and deploying the indoor navigation application. The TurtleBot 3 will be used for or mapping tasks, enabling the creation of indoor navigation maps.

| Description | Specifications |
|---|---|
| Model | MSI Laptop |
| Processor | Intel Core i5-12500H |
| Operating System | Windows 11 Home |
| Graphic | NVIDIA GeForce RTX 3050 4GB GDDR6 |
| Memory | 16GB DDR5 RAM |
| Storage | 512GB SSD |

*Table 5.1.1.1 Specifications of laptop*

| Description | Specifications |
|---|---|
| Model | Oppo Reno7 Z |
| Processor | Qualcomm Snapdragon 695 5G Octa-core |
| Operating System | Android 13 |
| Memory | 8GB |
| Storage | 128GB |

*Table 5.1.1.2 Specifications of Android phone*

| Description | Specifications |
|---|---|
| Model | TurtleBot 3 |
| Processor | Broadcom BCM2711 Quad-core Cortex-A72 (1.5GHz) |
| Operating System | Raspberry Pi OS (64-bit) |
| Memory | 2GB |
| Storage | 16GB microSD |

*Table 5.1.1.3 Specifications of TurtleBot 3*

### 5.1.2 Software Specifications

This section outlines the software components such as tools, frameworks and programming languages that involved in this project. They were used to support mobile application development, model training and other related tasks.

**Software Used:**

1. **Android Studio**

Android Studio is an integrated development environment (IDE) for Android app development that created and maintained by Google. It can be used to develop Flutter applications through official plugin support. In addition, Android Studio provides software development kit (SDK) integration, debugging utilities, and device or emulator management for efficient mobile application development.

2. **Github**

GitHub is a web-based platform that serves as a version control and code management tool. It enables developers to store, manage, and collaborate on software projects using Git. The central repository allows multiple branches to be maintained for experimenting with new features. Project code can then be merged across versions or reverted to earlier states when necessary.

3. **Oracle VirtualBox**

Oracle VirtualBox is a virtual machine that enables users to run multiple type of operating system within a supported virtualization environment. In this project, it was used to run Ubuntu 22.04 version it was used to run Ubuntu 22.04, which provided the necessary environment to configure the TurtleBot 3. At the same time, the virtual machine was also utilised for mapping tasks. This virtual machine allows configuration and navigation development to be performed within a stable and reliable environment.

4. **Kaggle**

Kaggle is a widely used online data science platform owned by Google. It provides access to notebooks, datasets, and machine learning resources. New learners can benefit from its learning resources to explore more about data science and machine learning. For this project, the relevant dataset was obtained from Kaggle to support model training.

5. **Roboflow**

Roboflow is a computer vision platform used for dataset management. This platform provides annotation tools that enable precise labelling of images, which is essential for training accurate models. In addition, Roboflow offers preprocessing and data augmentation features to improve dataset quality. It also allows datasets to be exported in multiple model formats with appropriate folder structures, label files, and metadata required for model training. In this project, the training dataset was obtained from Roboflow, with additional images collected and annotated to further enhance dataset quality and model performance.

6. **Google Colab**

Google Colab is a cloud-based Jupyter Notebook service offered by Google at no cost. It provides an accessible environment for writing and executing Python code through any web browser. By offering access to strong GPU and TPU resources, Google Colab is particularly suitable for users lacking advanced local hardware. This enables users to perform model training effectively. For this project, Google Colab was used to train the YOLOv8 model, which allows efficient model tunning and performance evaluation in a cloud environment.

**Programming Language and Framework**

1. **Flutter**

Flutter is an open-source framework developed by Google that enables the development of natively compiled applications for mobile, web, and desktop. It uses the Dart programming languages and has the capability to deploy applications across multiple platforms, including iOS, Android, web, macOS, Windows and Linux. In this project, Flutter was used to develop both the functionality and user interface of the proposed indoor navigation mobile application.

2. **Dart**

Dart is a popular programming language for building web, mobile, server, and desktop applications. It is designed to be easy to use and flexible, supporting deployment across multiple platforms.

3. **Python**

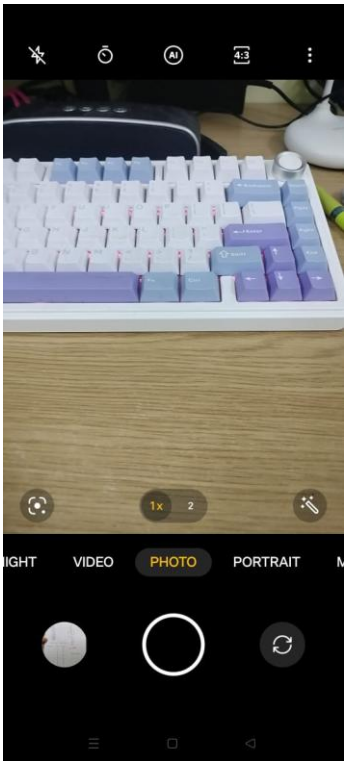
Python is a high-level programming language that widely used for machine learning and data processing. It offers extensive libraries and frameworks that support rapid development and experimentation. n this project, Python was utilised for training and testing the YOLOv8 object detection model, as well as handling data preprocessing and evaluation tasks.

**4. Ultralytics**

Ultralytics is a deep learning library that implements YOLO object detection models. It provides a high-level interface for training, validation, and simplifying computer vision development. Withtin this project, Ultralytics was used to train and fine-tune the YOLOv8n model with custom datasets, enabling efficient training, tuning and performance evaluation. The library also integrates seamlessly with Python environments such as Google Colab, supporting rapid experimentation and deployment.

## 5.2 Hardware Setup

During the preparation of this project, the mobile phone undergoes a comprehensive checkup, since the final deliverable will be deployed on the device. The camera, speaker, screen, and IMU sensors are tested to ensure proper functionality and to prevent issues during the development phase. According to Figure 5.2.1 and Figure 5.2.2, all components are confirmed to be operating correctly.



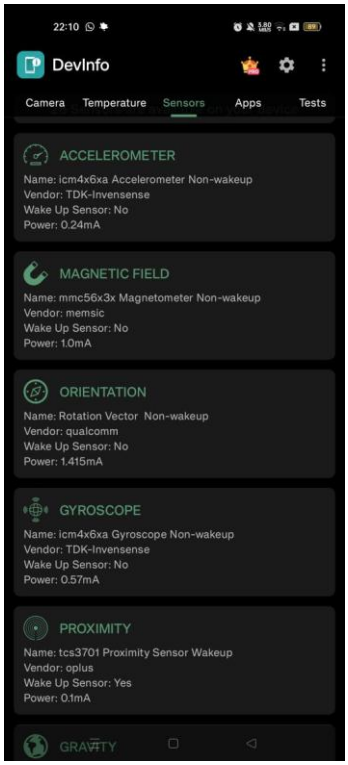




*Figure 5.2.1 Phone Camera Well Functions  Figure 5.2.2 IMU Sensor Well Functions*

## 5.3 Software Setup

Before starting the project, several software tools must be installed and relevant accounts must be created to support system development.

1. **Android Studio Otter**

Android Studio Otter (2025.2.1) is the primary development environment used to build the mobile application. It supports Flutter development through plugins and provides essential tools such as SDK integration, debugging features, and device emulation.

1. **Roboflow**

This platform serves as a source for datasets used in model learning and training. At the same time, its annotation feature allows users to label data for improved model accuracy.

2. Kaggle

This is the secondary source for obtaining datasets for model training and evaluation.

3. Github

It is used for version control and source code management. It enables tracking of changes as well as helps maintain and monitor the progress of the project.

4. Google Colab

This is the platform that used to train the YOLOv8n model. By providing a cloud-based environment with GPU support, it enables efficient training and rapid experimentation of models.

## 5.4 Module Implementation

The following section describes the implementation of the system's module. Each module is essential to the application's functionality. The core components covered here include the vision module, navigation module, auditory feedback module, and gesture input.

### 5.4.1 Vision Module

The implementation of the vision module is divided into several phrases including data preparation, data annotation, data extraction and preprocessing, label remapping, model training and model exporting. These phrases serve as the fundamental pipeline to successfully train the YOLOv8n object detection model.

**Data Preparation**

The data used in this project are obtained from Kaggle datasets, Roboflow, and manually captured images from the camera. The sources of these datasets are summarised in the Table 5.4.1.1.

| Classes | Sources |
|---|---|
| Table | Roboflow and manual captured images |
| Chair | Roboflow and manual captured images |
| Human | Kaggle |
| Door | Roboflow and manual captured images |
| QR code | Roboflow |

*Table 5.4.1.1 Sources of Datasets*

**Data Annotation**

For the images that were captured manually from the camera, the data annotation is needed to provide the YOLOv8n model with information about the target objects. Based on Figure 5.4.1.1, Roboflow's annotation features were used to support the precise drawing of bounding boxes and the accurate labeling of the target object in the images.


*Figure 5.4.1.1 Roboflow Annotation Features*

**Data Extraction and Preprocessing**

Since the datasets were generated in YOLOv8 format during both the preparation and annotation stages, they were inherently compatible with the YOLOv8n architecture. Therefore, no additional structural adjustments were required. However, the number of samples across classes was imbalanced. To prevent majority classes from dominating the model's learning process, controlled data extraction with random sampling was implemented based on Figure 5.4.1.2. Instead of exporting the entire dataset, this process capped the samples of each class to an equal number. Table 5.4.1.2 illustrates the exact number of samples extracted for the training and validation sets across each class. As a result, each class have 350 samples and organised into their respective directories. Altogether, the dataset now consists of 1,750 samples.

```
# Retrieve all image files
all_images = [f for f in os.listdir(src_img_dir) if f.endswith(('.jpg', '.png', '.jpeg'))]
actual_samples = min(len(all_images), num_samples)
if len(all_images) < num_samples:
    print(f" Extracting all images in {split_name} ({len(all_images)}) as it is less than the target")

# Random sampling
selected_images = random.sample(all_images, actual_samples)
```

*Figure 5.4.1.2 Code Snippet of Data Extraction*

| Classes | Training Set | Validation sets | Total number of extractions |
|---|---|---|---|
| Table | 250 | 100 | 350 |
| Chair | 250 | 100 | 350 |
| Human | 250 | 100 | 350 |
| Door | 250 | 100 | 350 |
| QR code | 250 | 100 | 350 |

*Table 5.4.1.2 Number of Training and Validation Samples per Class*

**Label Remapping**

Before model training, label remapping was required to prevent classification confusion. All datasets have different and overlapping class indexes since all of them are obtained from various sources. The initial class indexes for the five different datasets are shown in Table 5.4.1.3.

| | Human | Chair | Table | Door | QR code |
|---|---|---|---|---|---|
| **0** | human | chair | table | door | QR_CODE |
| **1** | - | CHAIR | - | - | qr_code |
| **2** | - | - | - | - | qrcode |

*Table 5.4.1.3 Initial Class Indexes in Each Class*

In the YOLO architecture, each label file uniquely associated to a specific image sample. These files consist of five numerical parameters: the class index, the x-coordinate of the bounding box center, the y-coordinate of the bounding box center, the bounding box width, and the bounding box height. To achieve label uniformity, the Python script in Figure 5.4.1.3 was executed. It started with opening each label file and reading the five variables one by one, with emphasis on the first variable (the class index). The original class index is evaluated against a predefined mapping dictionary and replaced with the unified index. The updated values are then written back to the file. After the execution of the python script, the new class indexes are created.

```python
for file_path in txt_files:
    with open(file_path, 'r') as f:
        lines = f.readlines()

    # Overwrite the file
    with open(file_path, 'w') as f:

        for line in lines:
            parts = line.strip().split()

            if parts:
                old_id = int(parts[0])

                if old_id in mapping:
                    parts[0] = str(mapping[old_id])
                f.write(" ".join(parts) + "\n")
```

*Figure 5.4.1.3 Code Snippet of Label Mapping*

**Dataset Directory Structure**

Based on Figure 5.4.1.4, all datasets are organized into two primary directories: train and valid. Each of these directories further contains two subdirectories, namely images and labels as shown in Figure 5.4.1.5 and Figure 5.4.1.6.

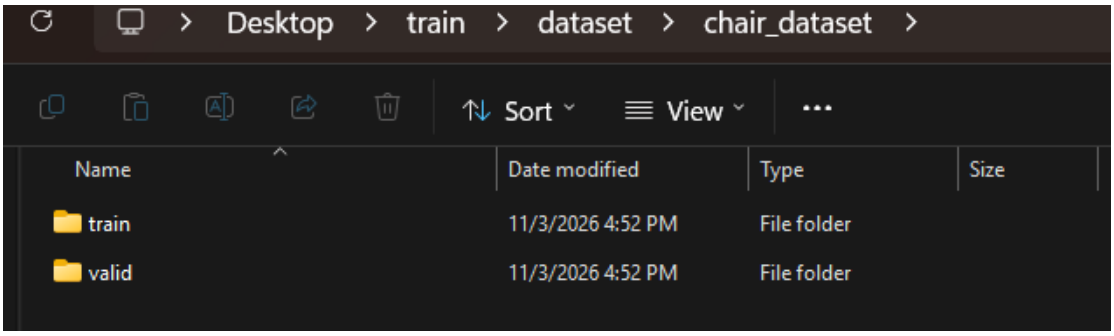



*Figure 5.4.1.4 Example of Chair Dataset Directory*

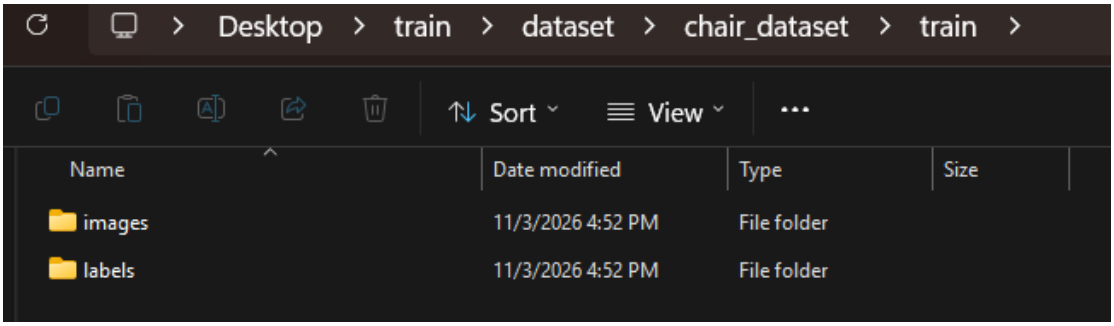



*Figure 5.4.1.5 Example of Chair Dataset's Train File Directory*

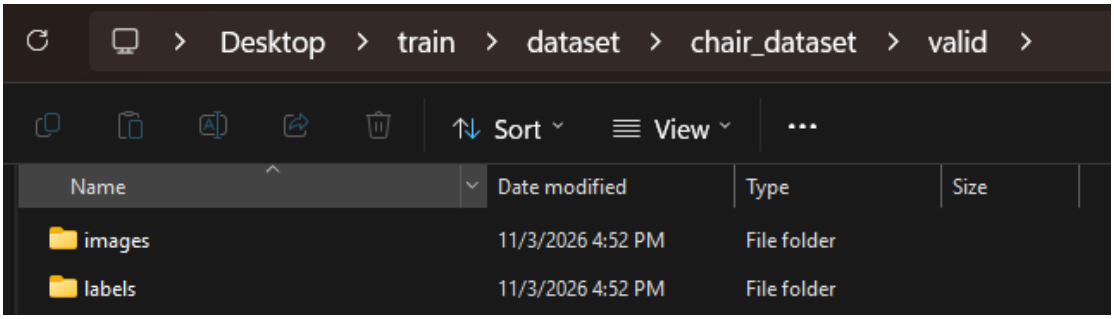



*Figure 5.4.1.6 Example Chair Dataset's Valid File Directory*

**Model Training**

For the model training phase, Google Colab was selected as the environment to train YOLOv8n model. First, all datasets were uploaded to Google Drive to ensure data persistence as Google Colab deletes all files in temporary storage once the runtime is disconnected. The code snippet in Figure 5.4.1.7 generates the required data.yaml configuration file. This script efficiently maps the training and validation paths directly to their respective dataset folders within the mounted Google Drive, rather than merging all datasets into a single directory. After installing ultralytics library, the code snippet that illustrated in Figure 5.4.1.8 was executed to initiate the YOLOv8n model training process.

```yaml
import yaml

# Define the absolute paths to the datasets
yaml_data = {
    'train': [
        '/content/drive/MyDrive/fyp/human_dataset/train/images',
        '/content/drive/MyDrive/fyp/chair_dataset/train/images',
        '/content/drive/MyDrive/fyp/table_dataset/train/images',
        '/content/drive/MyDrive/fyp/door_dataset/train/images',
        '/content/drive/MyDrive/fyp/QRcode_dataset/train/images'
    ],
    'val': [
        '/content/drive/MyDrive/fyp/human_dataset/valid/images',
        '/content/drive/MyDrive/fyp/chair_dataset/valid/images',
        '/content/drive/MyDrive/fyp/table_dataset/valid/images',
        '/content/drive/MyDrive/fyp/door_dataset/valid/images',
        '/content/drive/MyDrive/fyp/QRcode_dataset/valid/images'
    ],
    'nc': 5,
    'names': ['Human', 'Chair', 'Table', 'Door', 'QRcode']
}

with open('/content/drive/MyDrive/fyp/data.yaml', 'w') as f:
    yaml.dump(yaml_data, f, sort_keys=False)

print("Data.yaml generated successfully")
```

*Figure 5.4.1.7 Code Snippet of data.yaml File Configuration*

```yaml
model = YOLO('yolov8n.pt')

results = model.train(
    data='/content/drive/MyDrive/fyp/data.yaml',
    epochs=50,
    imgsz=640,
    batch=16,
    project='/content/drive/MyDrive/fyp/runs',
    name='fyp_model',
    workers=2,
    lr0=0.001,
    optimizer="Adam",
    patience=15
)
```

*Figure 5.4.1.8 Code Snippet of Model Training*

Some hyperparameters were customized during training, while the others remain unchanged.

**Hyperparameters Configuration:**

1. **data:** Indicates the path to the data.yaml file, which defines the directories for the training and validation sets.
2. **epochs:** Specifies the number of complete passes through the training dataset as 50.
3. **imgsz:** Defines the input image resolution that matches the optimal requirements of the YOLOv8n model, which is 640 x 640 pixels.
4. **batch:** Defines the number of samples processed in each forward and backward. A batch size of 16 was selected to optimize GPU memory usage in Google Colab while ensuring stable training performance.
5. **project and name:** Define the output directory. All training results are automatically stored under the [project]/[name] folder for easy retrieval.
6. **workers:** Define the number of CPU threads allocated as 2.
7. **lr0:** Determine the initial learning rate as 0.001.
8. **optimizer:** Define the optimization algorithm for this model training process. The Adam optimizer was selected due to its faster learning rate and more stable training performance compared to other optimization methods.
9. **patience:** Define the early stopping mechanism to halt the training process when the validation metrices show no improvements. It was set to 16.

**Model Exporting**

After model training phrases, the optimized YOLOv8n weights were saved as a PyTorch file. However, this format is not suitable for mobile environments. Therefore, the model was exported to TensorFlow Lite format with the code snippet in Figure 5.4.1.9.

```
model = YOLO(model_path)

export_result = model.export(
    format='tflite',
    optimize=True
)
```

*Figure 5.4.1.9 Code Snippet of Model Exporting*

**QR Code Decoder**

To enable QR code decoding, the Flutter plugin, mobile_scanner as shown in Figure 5.4.1.10 was installed. This decoder operates concurrently during user navigation. Once a QR code is successfully detected and decoded, the extracted coordinates are passed to the navigation module to override the VI user's current position.

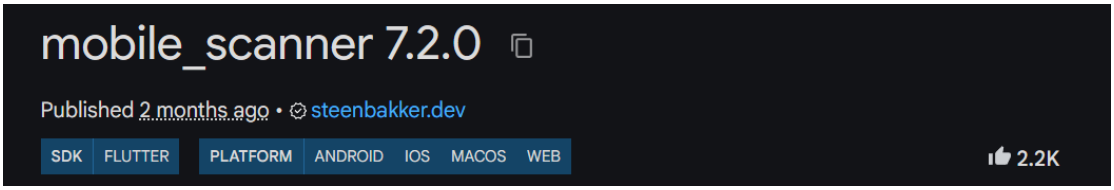



*Figure 5.4.1.10 mobile_scanner Flutter Plugin*

**Distance Estimation**

To estimate the distance between the user and the detected objects, the system implemented a distance estimation mechanism based on the principle of similar triangles. This approach was utilised because most of the smartphones rely on a single monocular camera. The system calculates the distance by comparing the known real height of the object in real world with the pixel height of its bounding box that generated on the screen. The formula is:

$$Distance = \frac{Real\ height\ of\ object\ \times Focal\ length\ of\ camera}{Pixel\ height\ of\ bounding\ box} \qquad (5.4.1.1)$$

According to the Figure 5.4.1.11, the average height of the object in real world are predefined within a dictionary. The system dynamically calculates the distance by

multiplying this predefined real height with the camera's focal length and dividing the pixel height of the bounding box.

```
class DistanceEstimator {
  static const Map<String, double> realHeights = {
    'person': 170.0,
    'chair': 75.0,
    'table': 75.0,
    'door': 200.0,
    'qrcode': 19,
  };

  static double calculate(String label, double pixelHeight, double focalLength) {
    double realH = realHeights[label.toLowerCase()] ?? 100.0;
    if (pixelHeight <= 0 || focalLength <= 0) return 0.0;
    return (realH * (focalLength)) / pixelHeight;
  }
}
```

*Figure 5.4.1.11 Code Snippet of Distance Estimation*

### 5.4.2 Navigation Module

#### Map Creation

To create a map for navigation purposes, the TurtleBot 3 Burger was configured to operate on Robot Operating System 2 (ROS 2) Humble. The robot performed the mapping process in the target indoor environment using SLAM (Simultaneous Localization and Mapping) technology with its LiDAR sensor. This process generated a two-dimensional occupancy grid map in .pgm format, which is illustrated in which is presented in Figure 5.4.2.1

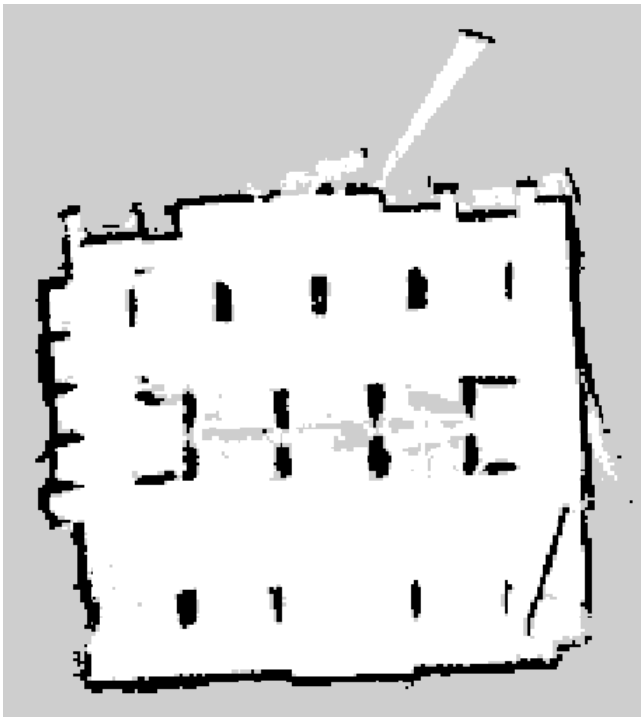


*Figure 5.4.2.1: Two-dimensional Occupancy Grid Map*

In the occupancy grid map, black areas represent obstacles, white areas represent free space, and grey areas indicate unknown regions. According to the Figure 5.4.2.2, the LiDAR sensor is mounted on the top of the TurtleBot 3. Due to the robot's height, the sensor can only scan objects at its scanning plane. As a result, objects that located above the LiDAR's height are not captured. Therefore, like table it only can scan the 4 legs of the table. This causes the decreasing in the overall quality and completeness of the generated map.

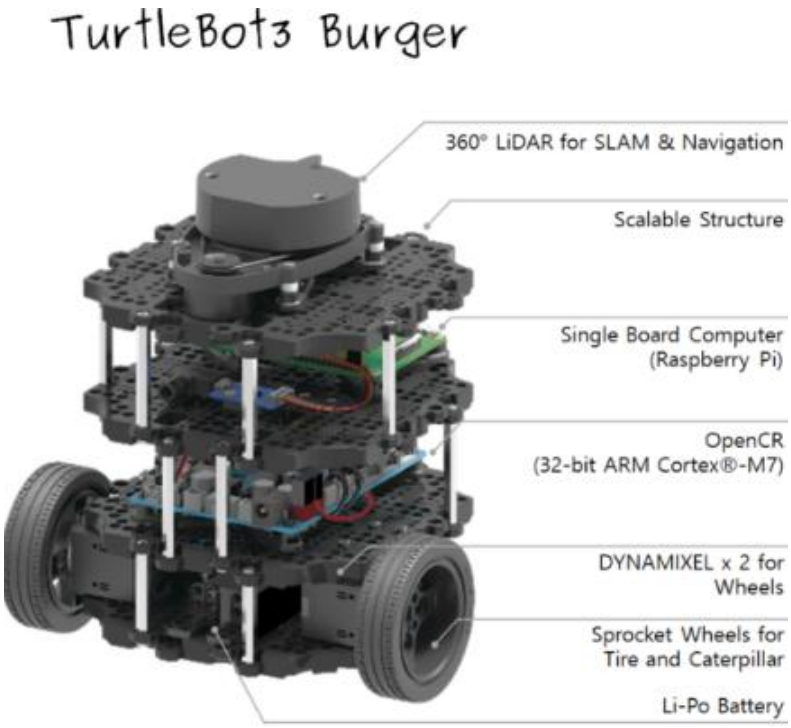



*Figure 5.4.2.2: TurtleBot 3 Burger Structure*

To improve the generated map, the occupancy grid needs to be updated by bounding the spaces that the LiDAR sensor is unable to capture. This helps to reduce the mapping errors by defining the inaccessible areas. The updated occupancy grid map is illustrated in as Figure 5.4.2.3.

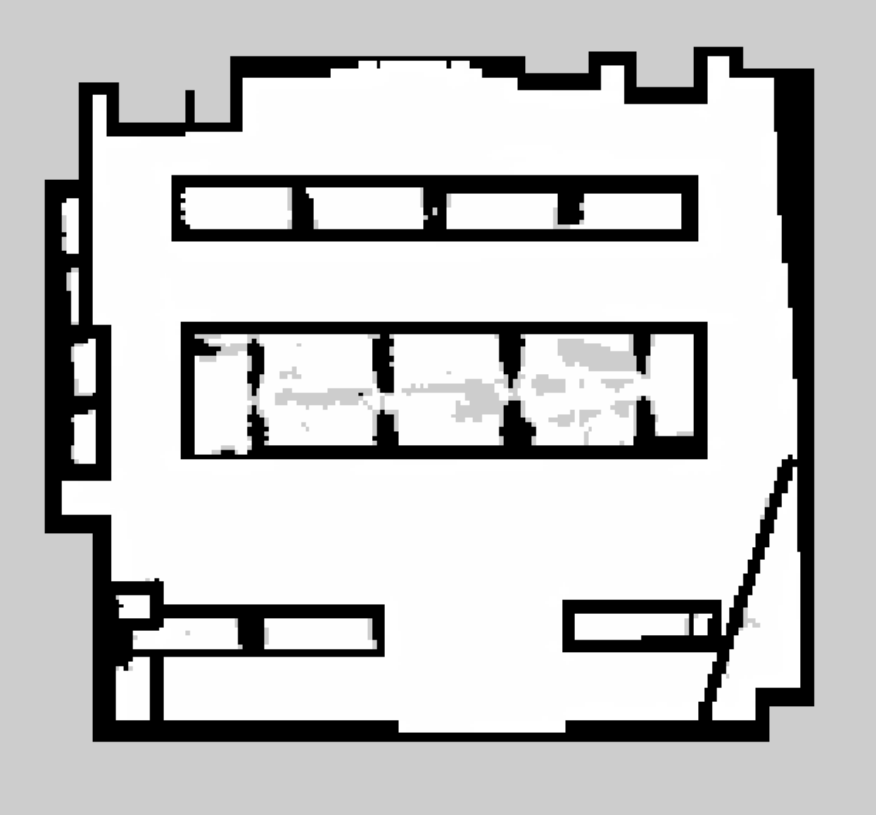


*Figure 5.4.2.3: Updated Two-dimensional Occupancy Grid Map*

**Device Sensor Collection**

To support the Pedestrian Dead Reckoning (PDR) in the system, the smartphone's IMU data must be collected. Thus, the Flutter package sensor_plus as shown in Figure 5.4.2.4 is required to access the device sensors. The SensorService class is responsible to manage accelerometer, gyroscope, and magnetometer data, which are essential for heading and step tracking. Based on the Figure 5.4.2.5, this class utilises asynchronous event listeners to continuously gather raw IMU data.

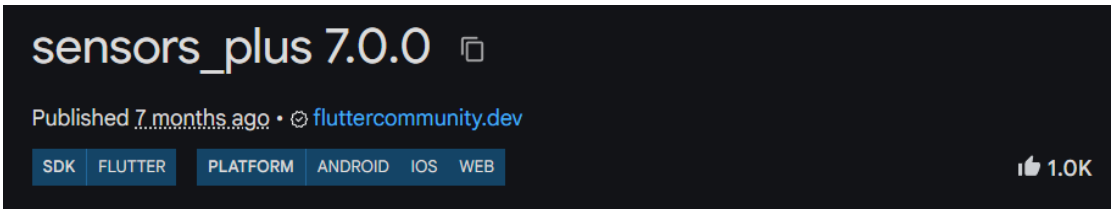



*Figure 5.4.2.4 sensor_plus Flutter Plugin*

```
Stream<SensorData> get sensorDataStream => _sensorDataController.stream;

List<double> _accelerometer = [0, 0, 0];
List<double> _gyroscope = [0, 0, 0];
List<double> _magnetometer = [0, 0, 0];

void startListening() {
  _accelerometerSub = accelerometerEventStream().listen((event) {
    _accelerometer = [event.x, event.y, event.z];
    _emitSensorData();
  });

  _gyroscopeSub = gyroscopeEventStream().listen((event) {
    _gyroscope = [event.x, event.y, event.z];
    _emitSensorData();
  });

  _magnetometerSub = magnetometerEventStream().listen((event) {
    _magnetometer = [event.x, event.y, event.z];
    _emitSensorData();
  });
}
```

*Figure 5.4.2.5 Code Snippet of Sensor Collect*

**Pedestrian Dead Reckoning**

1. **Step Detection**

The step detection is implemented with using the device sensor data, which are accelerometer and gyroscope. It processes these sensor readings to identify user movement in real time.

To make step detection more accurate, the system computes the square root of 3 axis acceleration reading. After that, gravity is subtracted from the magnitude to isolate the user's linear movement.

$$Total\ magnitude = \sqrt{x^2 + y^2 + z^2} - 9.81 \qquad (5.4.2.1)$$

A buffer mechanism is implemented to collect the continuous streams of sensor data. This mechanism allows to the system to get the average of the sensor data and smoothing out high-frequency noise. To ensure all counted steps are valid, some validation filters were applied.

- **Cooldown Period:** Implement a minimum time step interval to prevent counting on the same steps
- **Rotation Filter**: Reject steps if the gyroscope magnitude indicates the user rotate on the same position
- **Vertical Acceleration Filter**: Ensures the acceleration over a baseline threshold to confirm actual movement

### 2. Heading Tracking

To track user direction during the navigation, a heading monitoring system was implemented. It utilised the accelerometer and magnetometer data to determine the pitch and roll by applying the till-compensation formula as shown in Figure 5.4.2.6. At the same time, a buffer mechanism also implemented in here to smooth the heading data

```
double _calculateHeadingWithTiltCompensation(SensorData data) {
  double ax = data.accelerometer[1];
  double ay = -data.accelerometer[0];
  double az = data.accelerometer[2];

  double mx = data.magnetometer[1];
  double my = -data.magnetometer[0];
  double mz = data.magnetometer[2];

  double accelMagnitude = sqrt(ax * ax + ay * ay + az * az);
  if (accelMagnitude == 0) return _currentHeading;

  ax /= accelMagnitude;
  ay /= accelMagnitude;
  az /= accelMagnitude;

  double roll = atan2(ay, az);
  double pitch = asin(-ax);

  double magX = mx * cos(pitch) + mz * sin(pitch);
  double magY = my * cos(roll) - mz * sin(roll);

  double heading = atan2(magY, magX) * 180 / pi;
  heading = (heading + 360) % 360;

  return heading;
}
```

*Figure 5.4.2.6 Code Snippet of Heading Tracking*

### 5.4.3 Auditory Feedback Module

The Auditory feedback module is built in a class call TTSService by using this Flutter plugin called flutter_tts as shown in Figure 5.4.3.1. It is responsible for translating all navigation instructions and system status information into continuous auditory feedback.

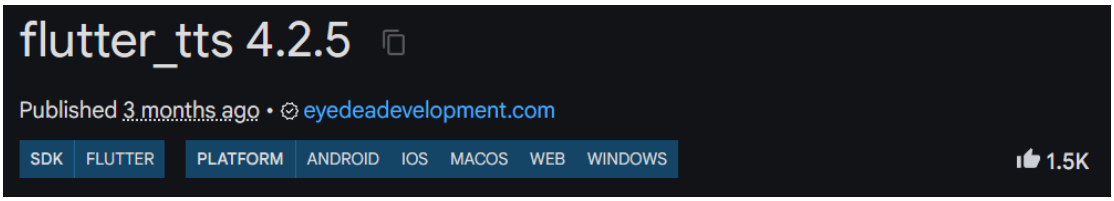



*Figure 5.4.3.1 futter_tts Flutter Plugin*

Based on Figure 5.4.3.2, all messages are categorised by priority. Critical messages, including obstacles warning had the highest priority that able to interrupt the current speech, while navigation instructions, such as “keep walk straight for 5 steps” had the lowest priority.

```
enum TTSPriority {
  LOW,
  NORMAL,
  HIGH
}
```

*Figure 5.4.3.2 Code Snippet of Message Priority*

At the same time, there were some TTS properties needed to be configured before using it. According to Figure 5.4.3.3, the language, speech rate, volume, and pitch level must be configured.

```dart
Future<void> _initTTS() async {
  await _flutterTts.setLanguage("en-US");
  await _flutterTts.setSpeechRate(0.5);
  await _flutterTts.setVolume(1.0);
  await _flutterTts.setPitch(1.0);

  _flutterTts.setCompletionHandler(() {
    _isSpeaking = false;
    _playNext();
  });

  _isInitialized = true;
  if (kDebugMode) print("TTS Service Initialized");
}
```

*Figure 5.4.3.3 Code Snippet of TTS Module*

### 5.4.4 Gesture Input

To maximize the accessibility for VI user, the application implemented non-visual gesture motion control to replace the conventional graphical user interface control.This design ensures that VI users can operate the application in a manner they are already familiar with.

**For Home screen:**

| Gesture Types | Action |
|---|---|
| Double taps | 1. Start to plan route<br>2. Confirm selection |
| Long press | Exit the application |
| Swipe left or right | 1. Switch to Navigation page or Recognition page<br>2. Browse the list of locations when configuring the start location and destination |

*Table 5.4.4.1 Type of Gesture Input in Home Screen*

**For Recognition screen and Navigation screen:**

| Gesture Types | Action |
|---|---|
| Long press | Stop current process and return to Home screen |

*Table 5.4.4.1 Type of Gesture Input in Recognition Screen and Navigation Screen*

## 5.5 System Operation

### 5.5.1 Splash Screen

This is the first interface that can be seen when launching the application. It acts as a loading page for the application as shown in Figure 5.5.1.1.

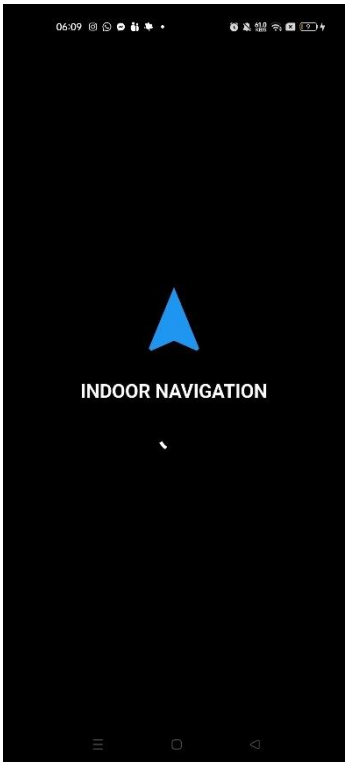



*Figure 5.5.1.1 Splash Screen*

### 5.5.2 Home Screen

After launching the application, the Home screen is displayed to the users. It serves as the entry point of the application. Based on Figure 5.5.2.1 and Figure 5.5.2.2, users can interact with the Navigation page or Recognition page to start using the feature of this application.

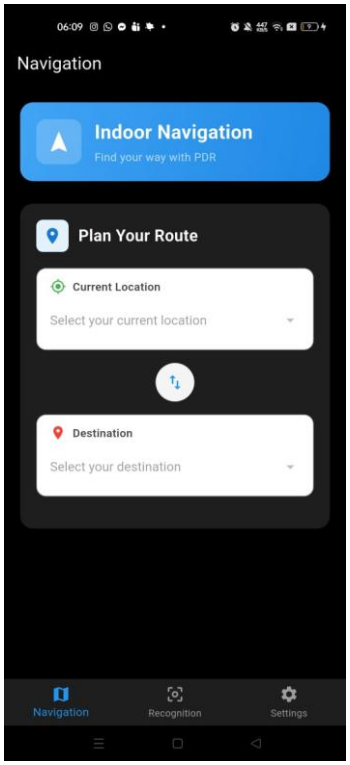



*Figure 5.5.2.1 Navigation Page of Home Screen*

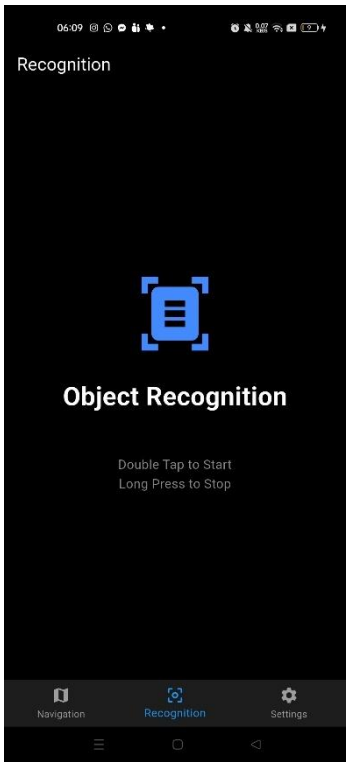



*Figure 5.5.2.2 Recognition Page of Home Screen*

### 5.5.3 QR Scanning Screen

On this interface, users can scan a QR code to override the system's current estimated location as shown in Figure 5.5.3.1. Users just need to simply face the phone camera to the QR code as the scanning process is executed automatically.

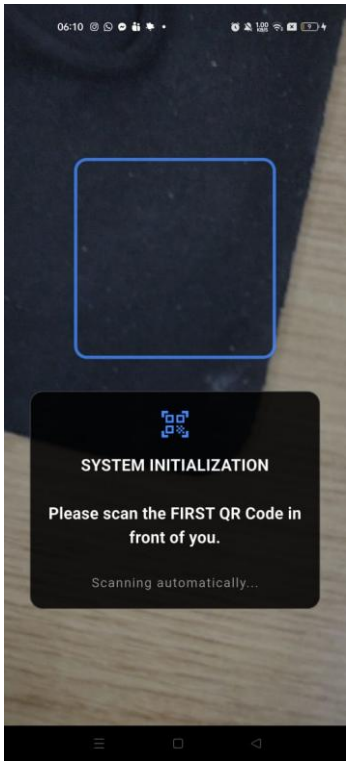



*Figure 5.5.3.1 QR Scanning Screen*

### 5.5.4 Navigation Screen

Once all device sensor as well as phone camera are activated and navigation path is calculated, users will be redirected to this screen as shown in Figure. All navigation details, including start location, destination, navigation instruction, user heading and step counts are displayed on the screen.


*Figure 5.5.4.1 Navigation Screen*

### 5.5.5 Recognition Screen

For Recognition screen, the system will display the details of the objects along with their estimated distance in the interface as shown in Figure 5.5.5.1.


*Figure 5.5.5.1 Recognition Screen*

## 5.6 Implementation Issues and Challenges

The major challenge encountered during implementation was the indoor navigation stage. Initially, Simultaneous Localization and Mapping (SLAM) was considered for deployment on mobile devices. However, this approach proved feasible only on high-end phones with Light Detection and Ranging (LiDAR) sensor and sufficient processing power. As an alternative, Visual Inertial Odometry (VIO) was researched, which fuses IMU data with camera input to estimate precise position and orientation in 3D space. I On Android devices, VIO is typically enabled through Google's ARCore framework, which provides the necessary camera–IMU synchronization and tracking capabilities. Nevertheless, the implementation could not proceed due to difficulties in running the ARCore plugin required for VIO integration. Therefore, Pedestrian Dead Reckoning (PDR) that utilises IMU data was adopted. While PDR provided a workable solution, it is not fully reliable due to accumulated errors and drift, highlighting the need for future improvements in accuracy and robustness.

Moreover, the creation of mapping for the system also presented significant issues. Due to limited prior experience, considerable time was required to complete the TurtleBot 3 setup, and multiple technical problems were encountered. These challenges were

eventually resolved with the guidance from Dr. Teoh, who has expertise in this area. At the same time, the mobile application development process was affected by limited experience. As a result, the development environment required repeated reinstallation and reconfiguration to restore functionality.

Furthermore, a problem was encountered when integrating the YOLO model with the QR code scanner. This is because both of them require continuous access to the camera frames as input. By implementing a cooldown mechanism, the system allows the YOLO model and the QR code scanner to take turns utilising the camera resources. This prevents the application from freezing issue.

## 5.7 Concluding Remark

This chapter present the development of the indoor navigation application. By using Flutter framework, the modules of the application were successfully implemented. Although there are some technical issues and challenges were faced during the development phrased, suitable alternatives were introduced. For instance, the PDR method was used to replace the VIO and SLAM methods and a cooldown mechanism was implemented to enable resource sharing between the YOLO model and the QR scanner. In conclusion, the overall development of the indoor navigation application is considered as success and ready to proceed to next phrase for testing and evaluation.

# CHAPTER 6
# System Evaluation and Discussion

## 6.1 System Testing and Performance Metrics

To evaluate the system's effectiveness, a comprehensive series of tests was conducted on the system. System testing is a critical phase that validates the complete, integrated system against its requirements. It examines the overall workflow and the interactions between modules to ensure the system functions as intended.

In this testing phrase, the black box functional testing is selected with using the use case testing techniques. This testing approach assesses the application's external behaviour by comparing the expected output to the actual output without considering the internal code. If the expected output matches the actual output, then the test case is validated as pass, which means the system successfully meets its requirements.

For the model evaluation, the performance of the YOLOv8n model was assessed using standard object detection metrics, which are Precision, Recall, and Mean Average Precision (mAP). These statistics metrics reflects the model ability to correctly identified the five predefined targets object: human, chair, table, door and QR code. The evaluation results provide a clear indication of the model's effectiveness in meeting the detection requirements of this project.

## 6.2 Testing Setup and Result

### 6.2.1 Use Case Testing

This section presents how the use case testing was conducted to validate the application's behaviour against the predefined use case. This testing process ensures that each scenario is executed as specified in order to confirm the system responds correctly to user interactions.

**Use Case Test 1 – Plan Route**

This use case test is related to the Use Case ID UC001.

| Test Coverage and Outcome | | | | |
|---|---|---|---|---|
| **Test Case ID** | **Test Condition /Dara** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| TC-01-001 | Main Flow – Plan Route | The system should allow VI user to select the start location and destination with non-visual interaction. The system also provides auditory feedback that includes initial usage instructions and real-time voice prompts for the currently focused locations. | The start location and destination are successfully inputted via gesture motion. At the same time, clear audio instructions are provided for the instruction of using Navigation page and currently focused location. | Pass |
| TC-01-002 | Alternative Flow – Navigate to Recognition page | The system should allow VI user to redirect to Recognition page in Home screen with non-visual interaction. | Navigation to the Recognition page of Home screen from the Navigation page is enabled by swiping left or right. | Pass |

*Table 6.2.2.1 Use Case Test 1*

**Use Case Test 2 – Start Navigation**

This use case test is related to the Use Case, UC002.

| Test Coverage and Outcome | | | | |
|---|---|---|---|---|
| **Test Case ID** | **Test Condition /Dara** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| TC-02-001 | Main Flow – Start Navigation | The system should start navigation by redirecting VI users to the Navigation Screen and activating all camera and device sensors. The system should require the user to scan two QR codes and calculate the shortest path to the destination before navigation begins. During navigation, the system shall track user steps and heading while simultaneously processing camera frames to detect obstacles and QR codes. When a QR code is detected, the system should override the user's current location. All | The system successfully redirected the user to the Navigation screen and activated all required camera and device sensors. It allows user to scan two QR codes and successfully calculated the shortest path to the destination. Throughout the navigation process, the system accurately tracked the user's steps and heading, while concurrently processing camera frames to detect obstacles. When detecting a QR code, the system successfully overrode the current user location. All navigation guidance delivered seamlessly | Pass |

| | | | | |
|---|---|---|---|---|
| | | navigation guidance and system status shall be announced through auditory feedback. | through auditory feedback. | |
| TC-02-002 | Alternative Flow – Detect Unknown QR Code On Start | The system should alert VI user to rescan the QR code if the current scan detects an unknown QR code. | When scanning an unknown QR code, the system notifies the user that the QR code is invalid and prompts them to rescan. | Pass |
| TC-02-003 | Alternative Flow – Detect Invalid Step | The system should be able to detect invalid user steps during navigation. | When the user turns around at the same place without taking any steps, the system ignores the current user step. | Pass |
| TC-02-004 | Alternative Flow – Detect Incorrect Heading | The system should be able to detect incorrect user heading and provide heading warning through audio cue during navigation | If the user's heading is incorrect, the system provides heading correction instructions through audio feedback. | Pass |
| TC-02-005 | Alternative Flow – Detect Unknown QR Code During Navigation | The system should ignore the unknown QR code during navigation. | During navigation, if the system detects any unknown QR codes, it does not update user current location. | Pass |
| TC-02-006 | Alternative Flow – Detect Not Last | If the current navigation node is not the last node, the | Before arriving at the destination, the system continuously updates | Pass |

| | | | | |
|---|---|---|---|---|
| | Navigation Node | system updates the Navigation screen details and continues to announce navigation guidance. | the navigation screen details and provides auditory feedback for the navigation guidance. | |

*Table 6.2.2.2 Use Case Test 2*

**Use Case Test 3 – Identify Object**

This use case test is related to the Use Case, UC003.

| **Test Coverage and Outcome** | | | | |
|---|---|---|---|---|
| **Test Case ID** | **Test Condition /Dara** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| TC-03-001 | Main Flow – Identify Object | The system should redirect VI user to Recognition screen and process camera frame to detect the objects. It also calculates the estimated distance to each detected object. For auditory feedback, the system provides instructions for using the Recognition Screen and announces the object name along with its estimated distance. The Recognition screen also supports | The system successfully redirected user to the Recognition screen and processed the camera frames in real-time to detect objects. The estimated distance to each detected object was calculated. Furthermore, the system successfully provided the auditory instructions for the screen and announced the correct object names along with their estimated distances. User able to control the Recognition screen | Pass |

| | | | | |
|---|---|---|---|---|
| | | gesture-based interaction. | with double taps and long press. | |
| TC-03-002 | Alternative Flow – Navigate to Navigation page | The system should allow VI user to redirect to Navigation page in Home screen with non-visual interaction. | Navigation to the Navigation page of Home screen from the Recognition page is enabled by swiping left or right. | Pass |

*Table 6.2.2.3 Use Case Test 3*

### 6.2.2 Model Evaluation

The evaluation in this section focuses on the YOLOv8n model's ability to accurately recognise the five predefined target objects: human, chair, table, door, and QR code. The key metrics that used to evaluate the model's performance are Precision (P), Recall (R), and Mean Average Precision (mAP). By analysing these quantitative metrics, the overall detection accuracy and reliability of the vision module can be validated.

**Key metrics:**

- **Precision:** Measures the percentage of the model's predictions that are correct.
- **Recall:** Measures of the model's ability to successfully find all actual target objects.
- **mAP@0.5:** The average precision across all classes calculated at a standard 50% overlap threshold.
- **mAP@0.5:0.95:** A stricter overall performance score averaged across multiple tighter overlap thresholds that are from 50% to 95%.

According to Figure 6.2.2.1, the the overall performance of the model achieved an acceptable Mean Average Precision of 79.3% across all classes and an overall Precision of 85.7%. This indicates that the model has the ability to make accurate predictions for the predefined target objects, which is a crucial safety factor for VI user. Looking at each class, the human class achieved the highest mAP of 97.4%. This is because human have distinct features that are easily differentiate from the background environment. Simultaneously, the chair class had the lowest Recall as 37.5% and overall mAP of

41.1%. As observed in the training dataset, each sample often contains multiple chairs together, causing them overlap or blocking each other. The model is accurate when a chair is completely visible, which achieved precision of 84.1%. However, its performance drops when multiple chairs are crowded together, leading to a lower recall. Since mAP is an overall score that balances precision and recall, this reduction in recall also causes the mAP to decrease.

```
Validating /content/drive/MyDrive/fyp/runs/fyp_model2/weights/best.pt...
Ultralytics 8.4.21 🚀 Python-3.12.12 torch-2.10.0+cu128 CUDA:0 (Tesla T4, 14913MiB)
Model summary (fused): 73 layers, 3,006,623 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all        500        733      0.857      0.752      0.793      0.629
                 Human        100        116      0.927      0.957      0.974       0.84
                 Chair         99        251      0.841      0.375      0.411      0.326
                 Table        100        118      0.798      0.705      0.791        0.5
                  Door        100        100      0.798      0.871      0.879      0.745
                QRcode        100        148      0.921      0.851      0.908      0.737
Speed: 0.2ms preprocess, 2.2ms inference, 0.0ms loss, 2.9ms postprocess per image
Results saved to /content/drive/MyDrive/fyp/runs/fyp_model2
```

*Figure 6.2.2.1 Performance of Model*

## 6.3 Project Challenges

Several challenges and limitations emerged throughout the development and testing stages. One of the most significant challenges faced was related to the indoor navigation part in the application. Although the system utilizes QR codes to update the user's current location, it still relies heavily on the device's built-in sensors for continuous tracking. This reliance causes drift errors to occur over time. As a result, the navigation becomes inaccurate and the estimated location will deviate from the VI user's actual physical position, especially if they miss scanning the QR code along the way.

Next, another challenge was the placement of the QR codes. If positioned too high or too low, the user would have to hold the device in an awkward and uncomfortable manner. Additionally, poor QR code placement will also affect the optimal scanning distance, causing the user to stand either too close or too far from the QR code. Moreover, environmental factors such as lighting could compromise the system's detection performance.

Furthermore, the application may experience brief freezing or latency due to shared camera resources and the limited processing power of the mobile device. This happens because both the QR code decoder and the YOLO model must process the live camera

feed simultaneously. Therefore, the system is forced to process the video frames sequentially, leading to a delay in performance.

## 6.4 Object Evaluation

In this chapter, the objectives of the project were evaluated to measure the effectiveness of the proposed application in addressing the needs of visually impaired users and achieving its intended goals. This evaluation was based on the testing outcomes and performance metrics that were determined during the system testing phase. This project aimed to develop an indoor navigation application for VI users.

The first objective was to provide a better and safe indoor navigation experience for visually impaired individuals, while the second objective was to design a lightweight system that is cost-effective and easy to use. Both objectives were successfully achieved, as the developed prototype operates on a standard smartphone without without requiring expensive external hardware. By integrating navigation and vision models, the application can calculate obstacle-free paths and guide visually impaired users to their desired destinations. Furthermore, the application is designed to be user-friendly, integrating non-visual gesture input so that VI users can interact with the system easily. Overall, the application delivers a highly accessible and affordable indoor navigation solution.

Additionally, the third objectives, integrate an object detection model for recognizing obstacles and objects was met by successfully fine-tuning and deploying a lightweight YOLOv8n model within the application. During testing, the model achieved an acceptable Mean Average Precision (mAP@0.5) of 79.3% in identifying the five predefined target objects. This allows the application to detect obstacles in real time, ensuring that VI users receive safe and continuous navigation support.

Moreover, the application delivers navigation guidance and system status information into audio form. This is essential for the VI users who lack of visual perception and cannot rely on the smartphone screen. By offering real-time auditory feedback, the application provides a smooth and satisfactory navigation experience to VI users.

## 6.5 Concluding Remark

In this chapter, a comprehensive and detailed testing and evaluation of the application was carried out. The black box functional testing demonstrated that the application able to execute path planning, navigation and object detection. In terms of model performance, the fine-tunned YOLOv8n model had successfully achieved a Mean Average Precision of 79.3%. This shows the reliability and stability of the model. However, there are some challenges such as sensor drift and latency were encountered during testing. The overall evaluation verifies that the application has met its defined objectives.

# CHAPTER 7
# Conclusion and Recommendation

## 7.1 Conclusion

This project was developed to address the challenges faced by VI individuals in navigating indoor environments, where GPS is ineffective. The motivation of this project comes from the fact that many existing assistive technologies are expensive and difficult to use. These factors reduce their accessibility for VI users and remain impractical for daily application. To overcome these issues, this project successfully proposed and developed a highly integrated, cost-effective indoor navigation and obstacle avoidance system tailored specifically for VI users.

The success of this project lies in the integration of three core modules: navigation module, vision model and auditory feedback module. The navigation module eliminates the reliance on costly indoor positioning hardware and technologies. By utilizing the smartphone's built-in sensors, it performs step detection and direction tracking algorithms to estimate the user's real-time position. Furthermore, the A* algorithm was implemented to compute the most optimal path, enabling users to navigate seamlessly from their starting point to their intended destination.

The vision module operates concurrently with the navigation module. It processes camera input using the YOLOv8n model to ensure user safety and positional accuracy. This module continuously scans the environment to detect physical obstacles during navigation. In addition, the model is trained to recognize specific QR codes, which serve as environmental checkpoints for recalibrating the user's precise location along the navigation path. Finally, the auditory feedback module acts as a bridge between the application and the VI user. By implementing this module, the need for visual interaction is eliminated. It delivers navigation instructions and obstacle warnings to the VI user through audio cues.

In conclusion, this project not only meets the project's initial objectives but also provides a foundation for an accessible, user-friendly indoor navigation solution. At the same time, it demonstrates how assistive technologies can be leveraged to improve the independence and safety of VI users.

## 7.2 Future Considerations and Recommendations

While the current system has successfully achieved its initial objectives, there are several key recommendations for future development to further enhance the performance, scalability, and overall user experience of this indoor navigation application.

Firstly, the current system navigation heavily relies on a pre-defined target environment map, which restricts application usability and limits its operation to specific environments. To improve system scalability, advanced technologies such as smartphone-based SLAM or ARCore localisation can be integrated, enabling the application to scan and map indoor environments in real time. This enhancement allows VI users to navigate safely even in unfamiliar environments. Simultaneously, these technologies also improve the accuracy of navigation that encountered in this project.

After that, integrating a dynamic path rerouting mechanism will further enhance the navigation capabilities of this application. Currently, the system utilizes the A* algorithm to calculate the optimal path, but it can be improved to react to environmental changes in real time. For example, the system should calculate a new route when the YOLOv8 model detects that the current navigation path is blocked. This would provide a more seamless and reliable navigation experience. At the same time, this reinforces user safety by ensuring continuous guidance even in changing environments.

Next, the current vision module is trained to detect a limited number of objects, including human, chair, table, door, and QR code. Although this is sufficient to demonstrate the proof of concept, real-world environments contain a wider range of objects. Therefore, future work should focus on expanding the dataset used for model training. This ensures a greater variety of detectable classes to enhance the safety and reliability of the obstacle avoidance feature in the application.

With the implementation of these recommendations, the indoor navigation application can become significantly more robust and reliable. These recommendations will empower VI users to move with greater confidence, safety, and independence in their daily lives.

# REFERENCES

[1] D. Khan, Z. Cheng, H. Uchiyama, S. Ali, M. Asshad, and K. Kiyokawa, “Recent advances in vision-based indoor navigation: A systematic literature review,” *Computers & Graphics*, vol. 104, pp. 24–45, May 2022, doi: https://doi.org/10.1016/j.cag.2022.03.005.

[2] W. Jeamwatthanachai, M. Wald, and G. Wills, “Indoor navigation by blind people: Behaviors and challenges in unfamiliar spaces and buildings,” *British Journal of Visual Impairment*, vol. 37, no. 2, pp. 140–153, May 2019, doi: https://doi.org/10.1177/0264619619833723.

[3] World Health Organization, “Blindness and vision impairment,” *world health organization*, Aug. 10, 2023. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment (Accessed: Aug. 25, 2025).

[4] JABATAN KEBAJIKAN MASYARAKAT, “Statistics on Persons with Disabilities (PWD,” Jun. 30, 2025. https://www.jkm.gov.my/main/documents/laporan-statistik-tahunan (Accessed: Aug. 25, 2025).

[5] J. Kunhoth, A. Karkar, S. Al-Maadeed, and A. Al-Ali, “Indoor positioning and wayfinding systems: a survey,” *Human-centric Computing and Information Sciences*, vol. 10, no. 1, May 2020, doi: https://doi.org/10.1186/s13673-020-00222-0.

[6] D. Plikynas, A. Žvironas, A. Budrionis, and M. Gudauskis, “Indoor Navigation Systems for Visually Impaired Persons: Mapping the Features of Existing Technologies to User Needs,” *Sensors*, vol. 20, no. 3, p. 636, Jan. 2020, doi: https://doi.org/10.3390/s20030636.

[7] A. Martinez-Sala, F. Losilla, J. Sánchez-Aarnoutse, and J. García-Haro, “Design, Implementation and Evaluation of an Indoor Navigation System for Visually Impaired People,” *Sensors*, vol. 15, no. 12, pp. 32168–32187, Dec. 2015, doi: https://doi.org/10.3390/s151229912.

[8] A. Idrees, Z. Iqbal, and M. Ishfaq, “An efficient indoor navigation technique to find optimal route for blinds using QR codes,” *IEEE Xplore*, Jun. 01, 2015. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7334197

[9] K. Gorro *et al.*, “Prototype of an Indoor Pathfinding Application with Obstacle Detection for the Visually Impaired,” *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 9, Jan. 2024, doi: https://doi.org/10.14569/ijacsa.2024.01509106.

[10] R. K. Katzschmann, B. Araki, and D. Rus, “Safe Local Navigation for Visually Impaired Users With a Time-of-Flight and Haptic Feedback Device,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, no. 3, pp. 583–593, Mar. 2018, doi: https://doi.org/10.1109/tnsre.2018.2800665.

[11] A. Gad *et al.*, “Safer Navigation: The AI-Powered Smart Cane for the Visually Impaired,” Dec. 2023, doi: https://doi.org/10.1109/dese60595.2023.10469180.

[12] S. M. Kumar, Vivek Vishnudas Nemane, R. Raman, N. Latha, and J. M. Shah, “IoT- BLE Based Indoor Navigation for Visually Impaired People,” pp. 444–448, Aug. 2023, doi: https://doi.org/10.1109/smarttechcon57526.2023.10391662.

[13] Vivek Kumar Paswan and A. Choudhary, “Camera Based Indoor Object Detection and Distance Estimation Framework for Assistive Mobility,” pp. 1–6, Dec. 2022, doi: https://doi.org/10.1109/soli57430.2022.10294458.

[14] P. J. Loresco, R. J. C. Cruz, K. Z. Ramones, J. A. D. Zafra, and K. R. G. Ramirez, “Indoor Navigation Glasses for the Visually Impaired with Deep Learning and Audio Guidance Using Google Coral Edge TPU,” *TENCON 2024 - 2024 IEEE Region 10 Conference (TENCON)*, pp. 842–845, Dec. 2024, doi: https://doi.org/10.1109/tencon61640.2024.10902929.

# APPENDIX A
# Code Sample

## A.1 Sensors Collector

```
import 'dart:async';
import 'package:sensors_plus/sensors_plus.dart';

class SensorData {
  final List<double> accelerometer;
  final List<double> gyroscope;
  final List<double> magnetometer;
  final DateTime timestamp;

  SensorData({
    required this.accelerometer,
    required this.gyroscope,
    required this.magnetometer,
    required this.timestamp,
  });

  SensorData copyWith({
    List<double>? accelerometer,
    List<double>? gyroscope,
    List<double>? magnetometer,
    DateTime? timestamp,
  }) {
    return SensorData(
      accelerometer: accelerometer ?? this.accelerometer,
      gyroscope: gyroscope ?? this.gyroscope,
      magnetometer: magnetometer ?? this.magnetometer,
      timestamp: timestamp ?? this.timestamp,
    );
  }
}

class SensorService {
  StreamSubscription? _accelerometerSub;
  StreamSubscription? _gyroscopeSub;
  StreamSubscription? _magnetometerSub;

  //send group of sensor data
  final StreamController<SensorData> _sensorDataController =
  StreamController<SensorData>.broadcast();

  Stream<SensorData> get sensorDataStream => _sensorDataController.stream;

  List<double> _accelerometer = [0, 0, 0];
  List<double> _gyroscope = [0, 0, 0];
```

```
  List<double> _magnetometer = [0, 0, 0];

  void startListening() {
    _accelerometerSub = accelerometerEventStream().listen((event) {
      _accelerometer = [event.x, event.y, event.z];
      _emitSensorData();
    });

    _gyroscopeSub = gyroscopeEventStream().listen((event) {
      _gyroscope = [event.x, event.y, event.z];
      _emitSensorData();
    });

    _magnetometerSub = magnetometerEventStream().listen((event) {
      _magnetometer = [event.x, event.y, event.z];
      _emitSensorData();
    });
  }

  //merge sensor data
  void _emitSensorData() {
    final data = SensorData(
      accelerometer: List<double>.from(_accelerometer),
      gyroscope: List<double>.from(_gyroscope),
      magnetometer: List<double>.from(_magnetometer),
      timestamp: DateTime.now(),
    );
    _sensorDataController.add(data);
  }

  void dispose() {
    _accelerometerSub?.cancel();
    _gyroscopeSub?.cancel();
    _magnetometerSub?.cancel();
    _sensorDataController.close();
  }
}
```

## A.2 Step Detector

```
import 'dart:async';
import 'dart:math';
import '../sensors/sensorsCollect.dart';

class StepData {
  final int totalSteps;
  final double movement;
  final DateTime timestamp;
```

```
  StepData({
    required this.totalSteps,
    required this.movement,
    required this.timestamp,
  });
}

class StepDetector {
  final StreamController<StepData> _stepController =
  StreamController<StepData>.broadcast();

  Stream<StepData> get stepStream => _stepController.stream;

  StreamSubscription? _sensorSubscription;


  static const double _gravity = 9.81;

  final List<double> _window = [];
  final int _windowSize = 10;
  final double _thresholdOffset = 0.35;
  final int _stepCooldownMs = 350;

  //vertical acceleration filter
  final List<double> _verticalAccelWindow = [];
  final int _verticalWindowSize = 5;
  final double _minVerticalAccel = 0.25;

  //rotation detection
  final List<double> _gyroMagnitudeWindow = [];
  final int _gyroWindowSize = 5;
  final double _maxGyroForStep = 1.5;

  //horizontal vs vertical movement ratio
  final List<double> _horizontalAccelWindow = [];
  final int _horizontalWindowSize = 5;
  final double _maxHorizontalVerticalRatio = 2.0;

  int _currentSteps = 0;
  DateTime _lastStepTime = DateTime.now();

  int _falsePositivesFiltered = 0;
  int _rotationFiltered = 0;
  int _scrollFiltered = 0;

  void startDetection(Stream<SensorData> sensorStream) {
    _sensorSubscription = sensorStream.listen((data) {
      _processAccelerometerReading(data);
```

```
  });
}

///process each accelerometer reading with multiple filters
void _processAccelerometerReading(SensorData data) {

 //calculate acceleration magnitude
 double magnitude = sqrt(
    data.accelerometer[0] * data.accelerometer[0] +
       data.accelerometer[1] * data.accelerometer[1] +
       data.accelerometer[2] * data.accelerometer[2]
 );

 double value = magnitude - _gravity;

 double verticalAccel = data.accelerometer[2].abs() - _gravity;
 _verticalAccelWindow.add(verticalAccel.abs());
 if (_verticalAccelWindow.length > _verticalWindowSize) {
  _verticalAccelWindow.removeAt(0);
 }

 double horizontalAccel = sqrt(
    data.accelerometer[0] * data.accelerometer[0] +
       data.accelerometer[1] * data.accelerometer[1]
 ) - _gravity;
 _horizontalAccelWindow.add(horizontalAccel.abs());
 if (_horizontalAccelWindow.length > _horizontalWindowSize) {
  _horizontalAccelWindow.removeAt(0);
 }

 double gyroMagnitude = sqrt(
    data.gyroscope[0] * data.gyroscope[0] +
       data.gyroscope[1] * data.gyroscope[1] +
       data.gyroscope[2] * data.gyroscope[2]
 );
 _gyroMagnitudeWindow.add(gyroMagnitude);
 if (_gyroMagnitudeWindow.length > _gyroWindowSize) {
  _gyroMagnitudeWindow.removeAt(0);
 }

 _window.add(value);
 if (_window.length > _windowSize) {
  _window.removeAt(0);
 }

 if (_window.length < _windowSize ||
    _verticalAccelWindow.length < _verticalWindowSize ||
    _horizontalAccelWindow.length < _horizontalWindowSize ||
    _gyroMagnitudeWindow.length < _gyroWindowSize) {
```

```
      return;
    }

    double avg = _window.reduce((a, b) => a + b) / _window.length;
    double dynamicThreshold = avg + _thresholdOffset;

    //calculate average values for filters
    double avgVertical = _verticalAccelWindow.reduce((a, b) => a + b) /
_verticalAccelWindow.length;
    double avgHorizontal = _horizontalAccelWindow.reduce((a, b) => a + b) /
_horizontalAccelWindow.length;
    double avgGyro = _gyroMagnitudeWindow.reduce((a, b) => a + b) /
_gyroMagnitudeWindow.length;

    //calculate horizontal/vertical ratio
    double hvRatio = avgVertical > 0.1 ? avgHorizontal / avgVertical : 999;

    //check cooldown
    DateTime now = data.timestamp;
    int timeSinceLastStep = now.difference(_lastStepTime).inMilliseconds;

    bool thresholdMet = value > dynamicThreshold;
    bool cooldownPassed = timeSinceLastStep > _stepCooldownMs;
    bool verticalAccelOk = avgVertical > _minVerticalAccel;
    bool notRotating = avgGyro < _maxGyroForStep;
    bool notScrolling = hvRatio < _maxHorizontalVerticalRatio;

    //info for rejected steps
    if (thresholdMet && cooldownPassed && !verticalAccelOk) {
      _scrollFiltered++;
    }

    if (thresholdMet && cooldownPassed && verticalAccelOk && !notRotating) {
      _rotationFiltered++;
    }

    if (thresholdMet && cooldownPassed && verticalAccelOk && notRotating
&& !notScrolling) {
      _falsePositivesFiltered++;
    }

    if (thresholdMet &&
        cooldownPassed &&
        verticalAccelOk &&
        notRotating &&
        notScrolling) {

      //detect valid step
      _currentSteps++;
```

```
      _lastStepTime = now;

      _stepController.add(StepData(
        totalSteps: _currentSteps,
        movement: value,
        timestamp: now,
      ));


    }
  }

  void stopDetection() {
    _sensorSubscription?.cancel();
  }

  void reset() {
    _currentSteps = 0;
    _window.clear();
    _verticalAccelWindow.clear();
    _horizontalAccelWindow.clear();
    _gyroMagnitudeWindow.clear();
    _lastStepTime = DateTime.now();
    _falsePositivesFiltered = 0;
    _rotationFiltered = 0;
    _scrollFiltered = 0;
  }

  void dispose() {
    _sensorSubscription?.cancel();
    _stepController.close();
  }

  int get currentSteps => _currentSteps;
  int get windowSize => _windowSize;
  double get thresholdOffset => _thresholdOffset;
  int get stepCooldown => _stepCooldownMs;

  Map<String, int> get filterStats => {
    'scrollFiltered': _scrollFiltered,
    'rotationFiltered': _rotationFiltered,
    'touchFiltered': _falsePositivesFiltered,
  };
}
```

## A.3 Heading

```
import 'dart:async';
import 'dart:math';
import '../sensors/sensorsCollect.dart'; import

class HeadingData {
  final double currentHeading;    // urrent direction (0-360°)
  final double roll;
  final double pitch;
  final DateTime timestamp;

  HeadingData({
    required this.currentHeading,
    required this.roll,
    required this.pitch,
    required this.timestamp,
  });

  // Get direction string (N, NE, E, etc.)
  String get cardinalDirection {
    if (currentHeading >= 337.5 || currentHeading < 22.5) return 'N';
    if (currentHeading >= 22.5 && currentHeading < 67.5) return 'NE';
    if (currentHeading >= 67.5 && currentHeading < 112.5) return 'E';
    if (currentHeading >= 112.5 && currentHeading < 157.5) return 'SE';
    if (currentHeading >= 157.5 && currentHeading < 202.5) return 'S';
    if (currentHeading >= 202.5 && currentHeading < 247.5) return 'SW';
    if (currentHeading >= 247.5 && currentHeading < 292.5) return 'W';
    if (currentHeading >= 292.5 && currentHeading < 337.5) return 'NW';
    return 'N';
  }
}

class HeadingService {
  final StreamController<HeadingData> _headingController =
  StreamController<HeadingData>.broadcast();

  Stream<HeadingData> get headingStream => _headingController.stream;

  StreamSubscription? _sensorSubscription;

  final List<double> _headingBuffer = [];
  final int _bufferSize = 5;
  double _currentHeading = 0.0;

  //start calculating heading from sensor stream
  void startTracking(Stream<SensorData> sensorStream) {
    _sensorSubscription = sensorStream.listen((data) {
      _processHeading(data);
    });
```

```
  }

  void _processHeading(SensorData data) {
    double rawHeading = _calculateHeadingWithTiltCompensation(data);
    double smoothedHeading = _applySmoothingBuffer(rawHeading);

    _currentHeading = smoothedHeading;

    _headingController.add(HeadingData(
      currentHeading: _currentHeading,
      roll: _calculateRoll(data.accelerometer),
      pitch: _calculatePitch(data.accelerometer),
      timestamp: data.timestamp,
    ));
  }

  double _calculateHeadingWithTiltCompensation(SensorData data) {
    double ax = data.accelerometer[1];
    double ay = -data.accelerometer[0];
    double az = data.accelerometer[2];

    double mx = data.magnetometer[1];
    double my = -data.magnetometer[0];
    double mz = data.magnetometer[2];

    double accelMagnitude = sqrt(ax * ax + ay * ay + az * az);
    if (accelMagnitude == 0) return _currentHeading;

    ax /= accelMagnitude;
    ay /= accelMagnitude;
    az /= accelMagnitude;

    double roll = atan2(ay, az);
    double pitch = asin(-ax);

    double magX = mx * cos(pitch) + mz * sin(pitch);
    double magY = my * cos(roll) - mz * sin(roll);

    double heading = atan2(magY, magX) * 180 / pi;
    heading = (heading + 360) % 360;

    return heading;
  }

  double _calculatePitch(List<double> accel) {
    return atan2(-accel[0], sqrt(accel[1] * accel[1] + accel[2] * accel[2]));
  }

  double _calculateRoll(List<double> accel) {
```

```
    return atan2(accel[1], accel[2]);
  }

  double _applySmoothingBuffer(double rawHeading) {
    _headingBuffer.add(rawHeading);
    if (_headingBuffer.length > _bufferSize) {
      _headingBuffer.removeAt(0);
    }
    if (_headingBuffer.length < _bufferSize) {
      return rawHeading;
    }

    double sumSin = 0;
    double sumCos = 0;
    for (double heading in _headingBuffer) {
      double radians = heading * pi / 180;
      sumSin += sin(radians);
      sumCos += cos(radians);
    }

    double avgRadians = atan2(sumSin / _bufferSize, sumCos / _bufferSize);
    double avgHeading = avgRadians * 180 / pi;
    return (avgHeading + 360) % 360;
  }

  void stopTracking() {
    _sensorSubscription?.cancel();
  }

  void dispose() {
    _sensorSubscription?.cancel();
    _headingController.close();
  }
}
```

## A.4 Recognition

```
import 'package:flutter/material.dart';
import 'package:ultralytics_yolo/ultralytics_yolo.dart';
import 'services/text_to_speech.dart';

class DistanceEstimator {
  static const Map<String, double> realHeights = {
    'person': 170.0,
    'chair': 75.0,
    'table': 75.0,
    'door': 200.0,
    'qrcode': 19,
```

```
  };

  static double calculate(String label, double pixelHeight, double focalLength) {
    double realH = realHeights[label.toLowerCase()] ?? 100.0;
    if (pixelHeight <= 0 || focalLength <= 0) return 0.0;
    return (realH * (focalLength + 25)) / pixelHeight;
  }
}

class RecognitionPage extends StatefulWidget {
  final dynamic cameraService;
  const RecognitionPage({super.key, required this.cameraService});

  @override
  State<RecognitionPage> createState() => _RecognitionPageState();
}

class _RecognitionPageState extends State<RecognitionPage> {
  final TTSService _ttsService = TTSService();
  bool _isDetecting = false;
  String _displayText = "Safe (scanning...)";

  DateTime _lastSpeakTime = DateTime.now();
  String _lastSpokenObject = "";

  @override
  void initState() {
    super.initState();
    Future.delayed(const Duration(milliseconds: 500), () {
      _ttsService.speak(
          "Object Recognition mode. Double tap to start detection. Long press to
stop.",
          priority: TTSPriority.HIGH
      );
    });
  }

  @override
  void dispose() {
    _ttsService.stop();
    super.dispose();
  }

  Future<void> _startDetection() async {
    _ttsService.speak("Starting camera, please wait.", priority: TTSPriority.HIGH);
    setState(() => _displayText = "Initializing Camera...");

    bool success = await widget.cameraService.initialize();
```

```
    if (success) {
      setState(() {
        _isDetecting = true;
        _displayText = "Safe (scanning...)";
      });
      _ttsService.speak("Camera started. Scanning environment.", priority:
TTSPriority.HIGH);
    } else {
      setState(() => _displayText = "Camera Error!");
      _ttsService.speak("Failed to start camera. Please check permissions.", priority:
TTSPriority.HIGH);
    }
  }

  void _stopDetection() {
    setState(() {
      _isDetecting = false;
      _displayText = "Safe (scanning...)";
      _lastSpokenObject = "";
    });
    _ttsService.speak("Detection stopped.", priority: TTSPriority.HIGH);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: SafeArea(
        child: GestureDetector(
          behavior: HitTestBehavior.translucent,
          onDoubleTap: () {
            if (!_isDetecting) _startDetection();
          },
          onLongPress: () {
            if (_isDetecting) {
              _stopDetection();
            } else {
              _ttsService.speak("Already stopped. Swipe left or right to change pages.",
priority: TTSPriority.HIGH);
            }
          },
          child: _isDetecting ? _buildDetectionView() : _buildStartView(),
        ),
      ),
    );
  }


  Widget _buildStartView() {
```

```
    return Container(
      width: double.infinity,
      color: Colors.black,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const Icon(Icons.document_scanner, size: 100, color: Colors.blueAccent),
          const SizedBox(height: 30),
          const Text(
            "Object Recognition",
            style: TextStyle(color: Colors.white, fontSize: 32, fontWeight:
FontWeight.bold),
          ),
          const SizedBox(height: 50),
          const Text(
            "Double Tap to Start\nLong Press to Stop",
            textAlign: TextAlign.center,
            style: TextStyle(color: Colors.white54, fontSize: 16, height: 1.5),
          ),
        ],
      ),
    );
  }

  Widget _buildDetectionView() {
    bool isSafe = _displayText == "Safe (scanning...)" || _displayText ==
"Initializing Camera...";
    Color statusColor = isSafe ? Colors.greenAccent : Colors.redAccent;

    return Stack(
      children: [
        YOLOView(
          modelPath: 'model/model+qr.tflite',
          task: YOLOTask.detect,
          onResult: (results) {
            if (results != null && results.isNotEmpty) {
              double closestDistance = 9999.0;
              String closestLabel = "";

              double hardwareFocalLength = widget.cameraService.focalLength;

              for (dynamic result in results) {
                String label = "unknown";
                try { label = result.className ?? result.label ?? "unknown"; } catch(e){}

                double pHeight = 0.0;
                try { pHeight = result.boundingBox?.height ?? result.rect?.height ??
0.0; } catch(e){}
```

```
            double distance = DistanceEstimator.calculate(label, pHeight,
hardwareFocalLength);

            if (distance > 0 && distance < closestDistance) {
              closestDistance = distance;
              closestLabel = label;
            }
           }

           if (closestDistance != 9999.0) {
            String newText = "Object: $closestLabel\nDistance:
${closestDistance.toStringAsFixed(1)} cm";
            if (_displayText != newText) {
              setState(() => _displayText = newText);
            }

            DateTime now = DateTime.now();
            int roundedDist = closestDistance.round();

            if (closestLabel != _lastSpokenObject ||
now.difference(_lastSpeakTime).inSeconds >= 3) {
              _ttsService.speak(
                 "$closestLabel, $roundedDist centimeters",
                 priority: TTSPriority.NORMAL
              );

              _lastSpeakTime = now;
              _lastSpokenObject = closestLabel;
            }
           }
          } else {
           if (_displayText != "Safe (scanning...)") {
            setState(() => _displayText = "Safe (scanning...)");
           }
          }
         },
        ),

        Positioned(
         bottom: 30,
         left: 20,
         right: 100,
         child: Container(
          padding: const EdgeInsets.symmetric(vertical: 15, horizontal: 10),
          decoration: BoxDecoration(
           color: Colors.black87,
           borderRadius: BorderRadius.circular(15),
           border: Border.all(color: statusColor, width: 2),
          ),
```

```
            child: Text(
              _displayText,
              style: TextStyle(
                color: statusColor,
                fontSize: 18,
                fontWeight: FontWeight.bold,
              ),
              textAlign: TextAlign.center,
            ),
          ),
        ),

        //stop button
        Positioned(
          bottom: 30,
          right: 20,
          child: FloatingActionButton(
            backgroundColor: Colors.redAccent,
            onPressed: _stopDetection,
            child: const Icon(Icons.stop, color: Colors.white, size: 30),
          ),
        ),
      ],
    );
  }
}
```

## A.5 Text to Speech

```
import 'package:flutter_tts/flutter_tts.dart';
import 'dart:async';
import 'dart:collection';
import 'package:flutter/foundation.dart';

enum TTSPriority {
  LOW,
  NORMAL,
  HIGH
}

class TTSMessage {
  final String text;
  final TTSPriority priority;
  final DateTime timestamp;

  TTSMessage(this.text, this.priority) : timestamp = DateTime.now();
}

class TTSService {
```

```
  static final TTSService _instance = TTSService._internal();
  factory TTSService() => _instance;

  final FlutterTts _flutterTts = FlutterTts();
  bool _isInitialized = false;
  bool _isSpeaking = false;

  final Queue<TTSMessage> _queue = Queue<TTSMessage>();

  String _lastSpokenText = "";
  DateTime _lastSpokenTime = DateTime.fromMillisecondsSinceEpoch(0);
  int _lastSpokenSteps = -1;

  final int _stepInterval = 3;
  final int _timeIntervalSeconds = 5;
  final int _emergencyCooldownSeconds = 2;

  TTSService._internal() {
    _initTTS();
  }

  Future<void> _initTTS() async {
    await _flutterTts.setLanguage("en-US");
    await _flutterTts.setSpeechRate(0.5);
    await _flutterTts.setVolume(1.0);
    await _flutterTts.setPitch(1.0);

    _flutterTts.setCompletionHandler(() {
      _isSpeaking = false;
      _playNext();
    });

    _isInitialized = true;
    if (kDebugMode) print("TTS Service Initialized");
  }

  void speak(String text, {TTSPriority priority = TTSPriority.LOW, int?
currentStep}) {
    if (!_isInitialized) return;
    DateTime now = DateTime.now();

    if (priority == TTSPriority.HIGH) {
      if (text == _lastSpokenText && now.difference(_lastSpokenTime).inSeconds <
_emergencyCooldownSeconds) return;

      _queue.clear();
      _flutterTts.stop();
      _isSpeaking = false;
      _queue.addFirst(TTSMessage(text, priority));
```

```
      _playNext();
      return;
    }

    if (priority == TTSPriority.NORMAL) {
      if (text == _lastSpokenText && now.difference(_lastSpokenTime).inSeconds <
3) return;
      _queue.clear();

      if (_isSpeaking && _lastSpokenText.contains("straight")) {
        _flutterTts.stop();
        _isSpeaking = false;
      }

      _queue.addFirst(TTSMessage(text, priority));
      if (!_isSpeaking) _playNext();
      return;
    }

    if (priority == TTSPriority.LOW) {
      bool shouldSpeak = false;

      if (now.difference(_lastSpokenTime).inSeconds >= _timeIntervalSeconds)
shouldSpeak = true;
      if (currentStep != null && _lastSpokenSteps != -1) {
        if ((currentStep - _lastSpokenSteps).abs() >= _stepInterval) shouldSpeak =
true;
      }
      if (text == _lastSpokenText && !shouldSpeak) return;

      if (shouldSpeak) {
        if (_queue.isNotEmpty) return;

        _queue.addLast(TTSMessage(text, priority));
        if (currentStep != null) _lastSpokenSteps = currentStep;
        if (!_isSpeaking) _playNext();
      }
    }
  }

  Future<void> _playNext() async {
    if (_queue.isEmpty || _isSpeaking) return;

    _isSpeaking = true;
    TTSMessage nextMsg = _queue.removeFirst();

    _lastSpokenText = nextMsg.text;
    _lastSpokenTime = DateTime.now();
```

```
    if (kDebugMode) print("Current TTS - [${nextMsg.priority.name}]:
${nextMsg.text}");
    await _flutterTts.speak(nextMsg.text);
  }

  void stop() {
    _queue.clear();
    _flutterTts.stop();
    _isSpeaking = false;
    _lastSpokenText = "";
    _lastSpokenSteps = -1;
  }
}
```

## A.6 Data_Extract

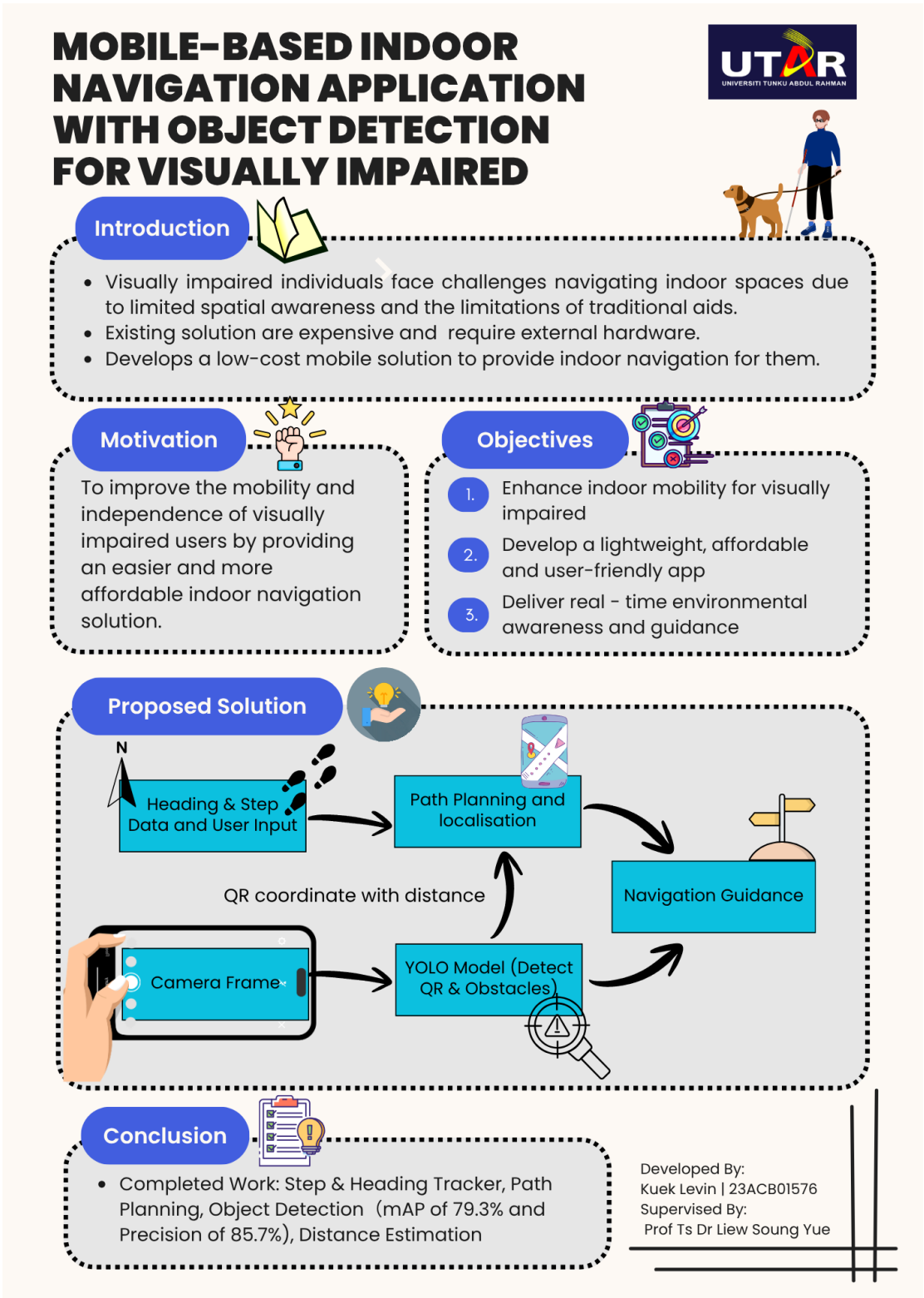
```
def extract_yolo_subset(source_main_dir, dest_main_dir, num_train=250,
num_valid=100):

    # Define the target splits and their corresponding extraction quantities
    splits = {
        'train': num_train,
        'valid': num_valid
    }

    for split_name, num_samples in splits.items():
        print(f"\n Processing [{split_name}] dataset, target extraction count:
{num_samples} images...")

        # Set source paths
        src_img_dir = os.path.join(source_main_dir, split_name, 'images')
        src_lbl_dir = os.path.join(source_main_dir, split_name, 'labels')

        # Check if source image directory exists
        if not os.path.exists(src_img_dir):
            print(f"Cannot find {src_img_dir}， Skipping extraction for {split_name}
dataset.")
            continue

        # Define destination paths
        dst_img_dir = os.path.join(dest_main_dir, split_name, 'images')
        dst_lbl_dir = os.path.join(dest_main_dir, split_name, 'labels')
        os.makedirs(dst_img_dir, exist_ok=True)
        os.makedirs(dst_lbl_dir, exist_ok=True)

        # Retrieve all image files
        all_images = [f for f in os.listdir(src_img_dir) if f.endswith(('.jpg', '.png',
'.jpeg'))]
        actual_samples = min(len(all_images), num_samples)
```

```
        if len(all_images) < num_samples:
            print(f" Extracting all images in {split_name} ({len(all_images)}) as it is
less than the target")

        # Random sampling
        selected_images = random.sample(all_images, actual_samples)

        success_count = 0
        missing_label_count = 0

        # Copy the selected images and their corresponding labels
        for img_name in selected_images:
            base_name = os.path.splitext(img_name)[0]
            lbl_name = base_name + '.txt'

            src_img_path = os.path.join(src_img_dir, img_name)
            src_lbl_path = os.path.join(src_lbl_dir, lbl_name)

            dst_img_path = os.path.join(dst_img_dir, img_name)
            dst_lbl_path = os.path.join(dst_lbl_dir, lbl_name)

            if os.path.exists(src_lbl_path):
                shutil.copy(src_img_path, dst_img_path)
                shutil.copy(src_lbl_path, dst_lbl_path)
                success_count += 1
            else:
                missing_label_count += 1

        print(f" [{split_name}] images {success_count} copied.")
        if missing_label_count > 0:
            print(f"{missing_label_count} images don't have corresponding labels and
were skipped.")

    # copy data.yaml if it exists
    src_yaml = os.path.join(source_main_dir, 'data.yaml')
    dst_yaml = os.path.join(dest_main_dir, 'data.yaml')
    if os.path.exists(src_yaml):
        shutil.copy(src_yaml, dst_yaml)
        print("\n data.yaml is copied to the destination dataset.")


SOURCE_DATASET = ' ‘
DESTINATION_DATASET = './door_dataset'
extract_yolo_subset(SOURCE_DATASET, DESTINATION_DATASET,
num_train=250, num_valid=100)
```

## A.7 Label_Remap

```
LABEL_MAPS = {
   'human_dataset': {0: 0},
   'chair_dataset': {0: 1, 1: 1, 2: 1, 3: 1, 4: 1},
   'table_dataset': {0: 2},
   'door_dataset': {0: 3},
   'QRcode_dataset': {0: 4, 1: 4, 2: 4}
}

for folder_name, mapping in LABEL_MAPS.items():
   # Dynamically locate all .txt label files within the respective dataset directory
   txt_files = glob.glob(f"./{folder_name}/**/labels/*.txt", recursive=True)

   # Initialize a counter to track the number of modified files
   count = 0

   for file_path in txt_files:
      with open(file_path, 'r') as f:
         lines = f.readlines()

      # Overwrite the file
      with open(file_path, 'w') as f:

         for line in lines:
            parts = line.strip().split()

            if parts:
               old_id = int(parts[0])d

               if old_id in mapping:
                  parts[0] = str(mapping[old_id])
               f.write(" ".join(parts) + "\n")
      count += 1

   print(f"[{folder_name}] Successfully updated {count} label files.")
```

# A.8 Poster


MOBILE-BASED INDOOR NAVIGATION APPLICATION WITH OBJECT DETECTION FOR VISUALLY IMPAIRED
UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Introduction
Visually impaired individuals face challenges navigating indoor spaces due to limited spatial awareness and the limitations of traditional aids.
Existing solution are expensive and require external hardware.
Develops a low-cost mobile solution to provide indoor navigation for them.
Motivation
To improve the mobility and independence of visually impaired users by providing an easier and more affordable indoor navigation solution.
Objectives
1. Enhance indoor mobility for visually impaired
2. Develop a lightweight, affordable and user-friendly app
3. Deliver real - time environmental awareness and guidance
Proposed Solution
N
Heading & Step Data and User Input
Path Planning and localisation
Navigation Guidance
QR coordinate with distance
Camera Frame
YOLO Model (Detect QR & Obstacles)
Conclusion
Completed Work: Step & Heading Tracker, Path Planning, Object Detection (mAP of 79.3% and Precision of 85.7%), Distance Estimation
Developed By:
Kuek Levin | 23ACB01576
Supervised By:
Prof Ts Dr Liew Soung Yue